68' MICRO JOURNAL

Australia	A $ 4.75	New Zealand	NZ $ 6.50
Singapore	S $ 9.45	Hong Kong	H $23.50
Malaysia	M $ 9.45	Sweden	30:-SEK

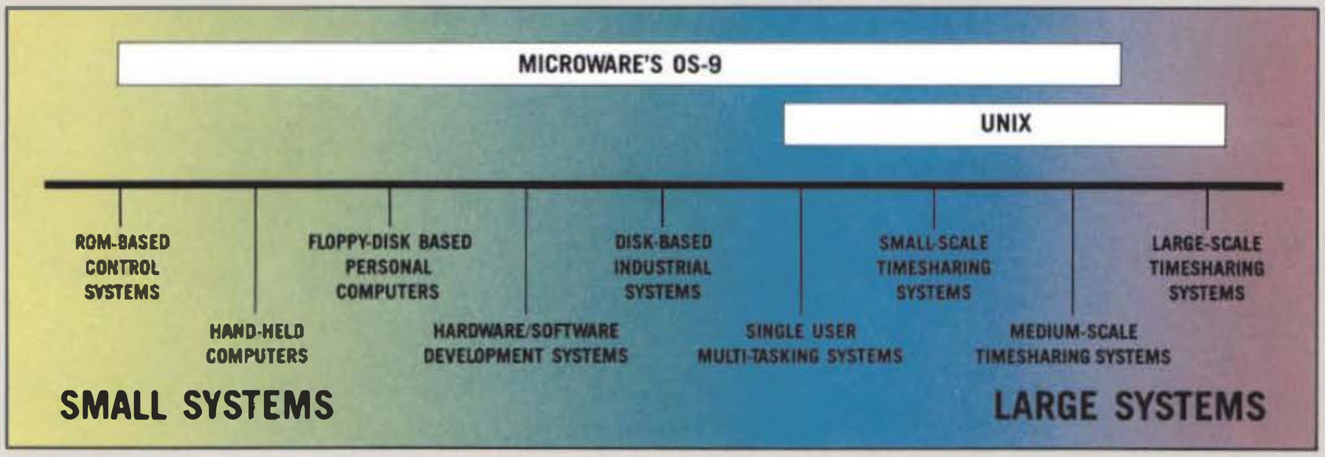
$2.95 USA

VOLUME VII ISSUE IX • Devoted to the 68XX User • September 1985

"Small Computers Doing Big Things"

PHOTO CREDIT: NASA

Only Microware's OS-9 Operating System Covers the Entire 68000 Spectrum


MICROWARE'S OS-9
UNIX
ROM-BASED CONTROL SYSTEMS
HAND-HELD COMPUTERS
FLOPPY-DISK BASED PERSONAL COMPUTERS
HARDWARE/SOFTWARE DEVELOPMENT SYSTEMS
DISK-BASED INDUSTRIAL SYSTEMS
SINGLE USER MULTI-TASKING SYSTEMS
SMALL-SCALE TIMESHARING SYSTEMS
MEDIUM-SCALE TIMESHARING SYSTEMS
LARGE-SCALE TIMESHARING SYSTEMS
SMALL SYSTEMS
LARGE SYSTEMS

Is complicated software and expensive hardware keeping you back from Unix? Look into OS-9, the operating system from Microware that gives 68000 systems a Unix-style environment with much less overhead and complexity.

OS-9 is versatile, inexpensive, and delivers outstanding performance on any size system. The OS-9 executive is much smaller and far more efficient than Unix because it's written in fast, compact assembly language, making it ideal for critical real-time applications. OS-9 can run on a broad range of 8 to 32 bit s stems based on the 68000 or 6809 family MPUs from ROM-based industrial controllers up to large multiuser s stems.

Key OS-9 Features At A Glance

- Compact (16K) ROMable executive written in assembly language
- User "shell" and complete utility set written in C
- C-source code level compatibility with Unix
- Full Multitasking/multiuser capabilities
- Modular design - extremely easy to adapt, modify, or expand
- Unix-type tree structured file system
- Rugged "crash-proof" file structure with record locking
- Works well with floppy disk or ROM-based systems
- Uses hardware or software memory management
- High performance C, Pascal, Basic and Cobol compilers

OS-9'S OUTSTANDING C COMPILER IS YOUR BRIDGE TO UNIX

Microware's C compiler technolog is another OS-9 advantage. The compiler produces extremely fast, compact, and ROMable code. You can easily develop and port s tem or application software back and forth to standard Unix s stems. Cross-compiler versions for VAX and PDP-11 make coordinated Unix/OS-9 software development a plea ure.

SUPPORT FOR MODULAR SOFTWARE — AN OS-9 EXCLUSIVE

Comprehensive support for modular software puts OS-9 a generation ahead of other operating s stems. It multiplies programmer productivity and memoi efficiency. Application software can be built from individually testable software modules including standard "librar " modules. The modular structure lets you customize and reconfigure OS-9 for specific hardware easily and quickly.

A SYSTEM WITH A PROVEN TRACK RECORD

Once an underground classic, OS-9 is now a solid hit. Since 1980 OS-9 has been ported to over a hundred 6809 and 68000 systems under license to some of the biggest names in the business. OS-9 has been imbedded in numerous consumer, industrial, and OEM products, and is supported b many independent software suppliers.

microware® OS-9

AUTHORIZED MICROWARE DISTRIBUTORS

AUSTRALIA
MICROPROCESSOR
CONSULTANTS
16 Bandera Avenue
Wagga Wagga 2650
NSW Australia
phone: 616-931-2331

ENGLAND
VIVAWAY LTD.
36-38 John Street
Luton, Bedfordshire
England LU1 2JE
phone: (0582) 423425
telex: 825115

JAPAN
MICROWARE JAPAN LTD.
3-8-9 Baraki, Ichikawa
Chiba 272-01, Japan
phone: 0473 (28) 4493
telex: 781-299-3122

SWEDEN
MICROMASTER
SCANDINAVIAN AB
S:t Persgatan 7
Box 1309
S751-43 Uppsala
Sweden
phone: 018-138595
telex: 76129

SWITZERLAND
ELSOFT AG
Bankstrasse 9
5432 Neuenhof
Switzerland
phone: (41) 056-862724
telex: 57136

USA
MICROWARE SYSTEMS
CORPORATION
1866 NW 114th Street
Des Moines, Iowa 50322
USA
phone: 515-224-1929
telex: 910-520-2535
FAX: 515-224-1352

WEST GERMANY
DR. KEIL, GMBH
Porphystrasse 15
D-6905 Schriesheim
West Germany
phone: (0 62 03) 67 41
telex: 465025

OS-9 is a trademark of Microware and Motorola. Unix is a trademark of Bell Labs.

ALL SYSTEMS INCLUDE:

- The CLASSY CHASSIS with a ferro-resonant, constant voltage power supply that provides + 8 volts at 30 Amps, + 16 volts a15 Amps, and − 16 volts at 5 Amps.
- Gold plated bus connectors.
- Double density DMA floppy disk controllers.
- Complete hardware and software documentation.
- Necessary cables, filler plates.

YOU CAN EXPAND YOUR SYSTEM WITH:

MASS STORAGE

Dual 8'' DSDD Floppies, Cabinet & Power Supply	$1698.88
20MB Streamer	(under development)
1.6MB Dual Speed Floppy	(under development)

MEMORY

#67 Static RAM-64K NMOS (6809 Only)	$349.67
#64 Static RAM-64K CMOS w/battery (6809 Only)	$398.64
#72 256K CMOS Static RAM w/battery	$998.72
#3116 Socket PROM/ROM/RAM Board (6809 only)	$268.31

INTELLIGENT I/O PROCESSOR BOARDS

significantly reduce systems overhead by handling routine I/O functions; freeing the host CPU for running user programs. This improves overall system performance and allows user terminals to be run at up to 19.2K baud. For use with GMX III and 020 systems.

#11 3 Port Serial-30 Pin (OS9)	$498.11
#14 3 Port Serial-30 Pin (UniFLEX)	$498.14
#12 Parallel-50 Pin (UniFLEX-020)	$538.12
#13 4 Port Serial-50 Pin (OS9 & UniFLEX-020)	$618.13

I/O BOARDS (6809 SYSTEMS ONLY)

#41 Serial, 1 Port	$88.41
#43 Serial, 2 Port	$128.43
#46 Serial, 8 Port (OS9/FLEX only)	$318.46
#42 Parallel, 2 Port	$88.42
#44 Parallel, 2 Port (Centronics pinout)	$128.44
#45 Parallel, 8 Port (OS9/FLEX only)	$198.45

CABLES FOR I/O BOARDS—SPECIFY BOARD

#95 Cable sets (1 needed per port)	$24.95
#51 Cent. B.P. Cable for #12 & #44	$34.51
#53 Cent. Cable Set	$36.53

OTHER BOARDS

#66 Prototyping Board-50 Pin	$58.66
#33 Prototyping Board-30 Pin	$38.33
Windrush EPROM Programmer S30 (OS9/FLEX 6809 only)	$545.00

CONTACT GIMIX FOR FURTHER DETAILS ON THESE AND OTHER BOARDS AND OPTIONS.

EXPORT MODELS: ADD $30 FOR 50Hz. POWER SUPPLIES.
ALL PRICES ARE F.O.B. CHICAGO.

GIMIX DOES NOT GUARANTEE PERFORMANCE OF ANY GIMIX SYSTEMS, BOARDS OR SOFTWARE WHEN USED WITH OTHER MANUFACTURERS PRODUCT.

GIMIX, Inc. reserves the right to change pricing, terms, and products specifications at any time without further notice.

	GIMIX 2MHZ 6809 SYSTEMS					GMX 68020 SYSTEMS	
Operating Systems Included	#49 OS9 GMX I/ and FLEX	#39 OS9 GMX II/ and FLEX	#79 OS9 GMX III/ and FLEX	#39 UniFLEX	#89 UniFLEX III	#020 OS9/68020	#020 UniFLEX VM
CPU Included	#05	#05	GMX III	#05	GMX III	GMX 020	GMX 020 + MMU
Serial Ports Included	2	2	3 Intelligent	2	3 Intelligent	3 Intelligent	3 Intelligent
High Speed Static RAM	64KB	256KB	256KB	256KB	256KB	512KB	1 Megabyte
PRICES OF SYSTEMS WITH: Dual 80 Track DSDD Drives	$2998.49	$3398.39	$4898.79	N/A	N/A	N/A	N/A
19MB Hard Disk and one 80 track Floppy Disk	$5598.49	$5998.39	$7798.79	$5998.39	$8098.89	$11.680.20	$13,680.20
72 MB Hard Disk and one 80 track	$7598.49	$7998.39	$9798.79	$7998.39	$10.098.89	$13.680.20	$15.680.20
a 72MB + a 6MB removable pack hard disk and one 80 track floppy	$9098.49	$9498.30	N/A	$9498.39	N/A	N/A	N/A
a 72MB + a 12MB removable pack hard disk and one 80 track floppy	N/A	N/A	$11.298.79	N/A	$11.598.89	$15.180.20	$17.180.20

GMX 6809 OS9/ FLEX SYSTEMS SOFTWARE			
OS9 + Editor, Assembler, Debugger	GMX I Included	GMX II Included	GMX III Included
FLEX	Included	Included	Included
GMXBUG Monitor	Included	Included	Included
Basic 09, RunB (OS9)	Included	Included	Included
RMS (OS9)	Included	Included	Included
DO (OS9)	Included	Included	Included
VDisk for FLEX	N/A	Included	Included
RAMDisk for OS9	N/A	$125 option	Included
O-FLEX	N/A	$250 option	Included
Support ROM	N/A	N/A	Included
Hardware CRC	N/A	N/A	Included

Available: Wide variety of languages and other software for use with either OS-9 or FLEX.

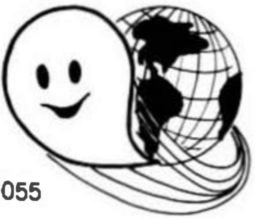
All GIMIX versions of OS9 can read and write RS color computer format OS9 disks, as well as the Microware/GIMIX standard format.

All OS9/FLEX systems allow you to software select either operating system.

TO ORDER BY MAIL: SEND CHECK OR MONEY ORDER OR USE YOUR VISA OR MASTER CHARGE. Please allow 3 weeks for personal checks to clear. U.S. orders add $5 handling if order is under $200.00. Foreign orders add $10 handling if order is under $200.00. Foreign orders over $200.00 will be shipped via Emery Air Freight COLLECT, and we will charge no handling. All orders must be prepaid in U.S. funds. Please note that foreign checks have been taking about 8 weeks for collection so we would advise wiring money, or checks drawn on a bank account in the U.S. Our bank is the Continental Illinois National Bank of Chicago, 231 S. LaSalle Street, Chicago, IL 60693, account number 73-32033.

BASIC-09 and OS-9 are trademarks of Microware Systems Corp. and MOTOROLA, Inc. FLEX and UniFLEX are trademarks of Technical Systems Consultants, Inc. GIMIX, GHOST, GMX, CLASSY CHASSIS, are trademarks of GIMIX, Inc.

GIMIX inc.



1337 WEST 37th PLACE
CHICAGO, ILLINOIS 60609
(312) 927-5510 • TWX 910-221-4055

'68' MICRO JOURNAL

Portions of the text for '68' Micro Journal were prepared using the following furnished Hard/Software:

COMPUTERS - HARDWARE

Southwest Techincal Products
219 W. Rhapsody
San Antonio, TX 78216
S09 - 5/8 DMF Disk - CDS1 - 8212W - Sprint 3 Printer

GIMIX Inc.
1337 West 37th Place
Chicago, IL 60609
Super Mainframe - OS9 - FLEX - Assorted Hardware

EDITORS - WORD PROCESSORS

Technical Systems Consultants, Inc.
111 Providence Road
Chapel Hill, NC 27514
FLEX - Editor - Text Processor

Great Plains Computer Co., Inc.
PO Box 916
Idaho Falls, ID 83401
Stylograph - Mail Merge - Spell

Editorial Staff

Don Williams Sr.	Publisher
Larry E. Williams	Executive Editor
Tom E. Williams	Production Editor
Robert L. Nay	Technical Editor

Administrative Staff

Mary Robertson	Office Manager
Penny Williams	Subscriptions
Christine Kocher	Accounting

Contributing Editors

Ron Anderson
Peter Dibble
Dr. Theo Elbert
Dr. E. M. Pass
Philip Lucido
Norm Commo
William E. Fisher
Carl Mann
Ron Voigts

Special Technical Projects

Clay Abrams K6AEP
Tom Hunt

CONTENTS

Vol. VIII, Issue VIII September 85

Send All Correspondence To:

Computer Publishing Center
68' Micro Journal
5900 Cassandra Smith Rd.
Hixson, Tn. 37343

Phone (615) 842-4600 or Telex 558 414 PVT BTH

68' Micro Journal is published 12 times a year by Computer Publishing Inc. Second Class Postage Paid ISSN 0194-5025 at Hixson, Tn. and additional entries. Postmaster: send form 3597 to 68' Micro Journal, POB 849 Hixson, Tn. 37343.

Subscription Rates

1 Year $24.50 U.S.A., Canada & Mexico Add $9.50 a Year. Other Foreign Add $12 a Year for Surface, Airmail Add $48 a Year. Must be in U.S. currency!

Items or Articles For Publication

Articles submitted for publication should include authors name, address, telephone number and date. Articles should be on either 5 or 8 inch disk in STYLOGRAPH or TSC Editor format with 3.5 inch Column width. All disks will be returned. Articles submitted on paper should be 4.5 inches in width (including Source Listings) for proper reductions. Please Use A Dark Ribbon!! No Blue Ink!!! Single space on 8X11 bond or better grade paper. No hand written articles accepted. Disks should be in FLEX2 6800 or FLEX9 6809 any version or OS-9 any version.

The following TSC Text Processor commands ONLY should be used: .sp space, .pp paragraph, .fi fill and .nf no fill. Also please do not format within the text with multiple spaces. We will enter the rest at time of editing.

All STYLOGRAPH commands are acceptable except .pg page command. We print edited text files in continous text form.

Letters To The Editor

All letters to the editor should comply with the above requirements and must be signed. Letters of "gripes" as well as "praise" are solicited. We reserve the right to reject any submission for lack of "good taste" and we reserve the right to define "good taste".

Advertising Rates

Commercial advertisers please contact 68' Micro Journal advertising department for current rate sheet and requirements.

Classified Advertising

All classified ads must be non-commercial. Minimum of $9.50 for first 20 words and .45 per word after 20. All classifieds must be paid in advance. No classified ads accepted over the phone.

..HEAR YE.....HEAR

OS-9™ User Notes

By: Peter Dibble
As Published in 68 Micro Journal

The publishers of 68 Micro Journal are proud to announce the publication of Peter Dibbles **OS9 USER NOTES**.

Information for the BEGINNER to the PRO,
Regular or CoCo OS9

Using OS9
HELP, HINTS, PROBLEMS, REVIEWS, SUGGESTIONS, COMPLAINTS, OS9 STANDARDS, Generating a New Bootstrap, Building a new System Disk, OS9 Users Group, etc.

Program interfacing to OS9
DEVICE DESCRIPTORS, DIRECTORIES, "FORKS", PROTECTION, "SUSPEND STATE", "PIPES", "INPUT/OUTPUT SYSTEM", etc.

Programming Languages
Assembly Language Programs and Interfacing; Basic09, C, Pascal, and Cobol reviews, programs, and uses; etc.

Disks Include
No typing all the Source Listings in. Source Code and, where applicable, assembled or compiled Operating Programs. The Source and the Discussions in the Columns can be used "as is", or as a "Starting Point" for developing your OWN more powerful Programs. Programs sometimes use multiple Languages such as a short Assembly Language Routine for reading a Directory, which is then "piped" to a Basic09 Routine for output formatting, etc.

BOOK Typeset -- w/ Source Listings (3-Hole Punched; 8 x 11) **$9.95**

Deluxe Binder - - - - - - - - - - - $5.50

All Source Listings on Disk
1 8" SS, SD Disk - - - - $14.95
2 5" SS, DD Disks - - - $24.95

Shipping and Handling; $3.50 per Book, $2.50 per Disk Set
* All Currency in U.S. Dollars   Foreign Orders Add $4.50 S/H
If paying by check - Please allow 4-6 weeks delivery

Continually Updated In 68 Micro Journal Monthly

Computer Publishing Inc.
5900 Cassandra Smith Rd.
Hixson, TN. 37343

VISA

(615) 842-4600
Telex 558 414 PVT BTH

TM - OS9 is a trademark of Microware Systems Corp. and Motorola Inc.
TM - 68' Micro Journal is a trademark of Computer Publishing Inc.

More help than any other thing I have ever read, except 68 Micro Journal, of course. Come on with the C book on Norm Commo's column.
BD

FLEX™ USER NOTES
THE 6800-6809 BOOK

By: Ronald W. Anderson
As published in 68 MICRO JOURNAL™

SWELL! - Best buy EVER in a 6800/6809 book.
JTW

The publishers of 68 MICRO JOURNAL are proud to announce the publication of Ron Anderson's **FLEX USER NOTES**, in book form. This popular monthly column has been a regular feature in 68 MICRO JOURNAL SINCE 1979. It has earned the respect of thousands of 68 MICRO JOURNAL readers over the years. In fact, Ron's column has been described as the 'Bible' for 68XX users, by some of the world's leading microprocessor professionals. Now all his columns are being published, in whole, as the most needed and popular 68XX book available. Over the years Ron's column has been one of the most popular in 68 MICRO JOURNAL. And of course 68 MICRO JOURNAL is the most popular 68XX magazine published.

As a SPECIAL BONUS all the source listing in the book will be available on disk for the low price of: FLEX™ format only — 5" $12.95 — 8" $16.95 plus $2.50 shipping and handling, if ordered with the book. If ordered separately the price of the disks will be: 5" $17.95 — 8" $19.95 plus $2.50 shipping and handling.

Listed below are a few of the **TEXT** files included in the book and on diskette.

All TEXT files in the book are on the disks.

LOGO.C1	File load program to offset memory — ASM PIC
MEMOVE.C1	Memory move program — ASM PIC
DUMP.C1	Printer dump program — uses LOGO — ASM PIC
SUBTEST.C1	Simulation of 6800 code to 6809, show differences — ASM
TERMEM.C2	Modem input to disk (or other port input to disk) — ASM
M.C2	Output a file to modem (or another port) — ASM
PRINT.C3	Parallel (enhanced) printer driver — ASM
MODEM.C2	TTL output to CRT and modem (or other port) — ASM
SCIPKG.C1	Scientific math routines — PASCAL
U.C4	Mini-monitor, disk resident, many useful functions — ASM
PRINT.C4	Parallel printer driver, without PFLAG — ASM
SET.C5	Set printer modes — ASM
SETBAS1.C5	Set printer modes — A-BASIC
	(And many more)

**Over 30 TEXT files included in ASM (assembler) — PASCAL — PIC (position independent code) TSC BASIC-C, etc.

NOTE: .C1,.C2, etc. = Chapter 1, Chapter 2, etc.

This will be a limited run and we cannot guarantee that supplies will last long. Order now for early delivery.

Finally I don't have to type in all that source code, a GREAT, GREAT idea. Thanks and good luck, you guys are the best!
TOA

Foreign Orders Add $4.50 S/H

Softcover — Large Format

Book only: **$7.95** + $2.50 S/H

With disk: 5" **$20.90** + $2.50 S/H

With disk: 8" **$22.90** + $2.50 S/H

I will keep this (FLEX USER NOTES) book always beside my computer. I have not ever, with any magazine, received as much information as this one book, and 68 MJ.
DIB

See your local S50 dealer/bookstore or order direct from:

Computer Publishing Inc.
5900 Cassandra Smith Rd.
Hixson, TN 37343
(615) 842-4601

TELEX 558 414 PVT BTH

™FLEX is a trademark of Technical Systems Consultants

FLEX User Notes

Ronald W. Anderson
3540 Sturbridge Court
Ann Arbor, Mi 48105

Editor Project

Last time I mentioned my "Great Editor" project. I have been busily working away at the project, and the PL/9 object code now is about 10K. I am hoping to keep it under 13K since I have about 3K of variables. That will leave a full 32K for the file buffer that holds the file being edited. If I can keep the buffer that big, I should be able to edit a file of some 125 sectors all at once. I spent about a week at this project, which, of course has put me almost full time back on the SWTPc system. At the end of the week, the editor was pretty buggy and I was weary. Progress was very slow, so I spent a week thinking about other things. The first day back on the project, I made some considerable progress eliminating bugs, and since then, I've spent another week at it, adding several more features.

At this point, I can "see the light at the end of the tunnel". That is, there is not a whole lot more to implement, though what is left is not trivial. I did decide that I needed a little break from the effort so I am writing this as a "relief".

More Feedback

I have received several more letters from readers since writing last month's column. Nearly all have indicated an interest in topics on FLEX and how it works. I'll continue inclusion of some such topic each month. A few letters had very specific questions. One reader wrote for advice on how to get OS-9 running on his SWTPc system. All I know about that is that it is necessary to use a GIMIX processor board and monitor chip to do it. The question would have been better addressed to Peter Dibble (maybe it was sent to him also). I'll write an answer, but it won't be of much help. Another reader wrote me a list of early bugs in FLEX and thought a little historical information might be of interest to many readers. I guess I was lucky not to have been tripped up by a couple of the bugs he mentioned, since I had one of the earliest versions of FLEX2. I had a pair of disk drives before FLEX, and when it was made available, I received a copy of it. This writer also mentioned one bug that would be of general interest, particularly to new users of FLEX, the problem of changing disks in a drive with a file or files open. (If the program you are running doesn't make special provision for doing this, DON'T. It will wreck your disk. I'll get into the reasons another time, since I seem to have used up my space for this month).

Aside from the general agreement about FLEX topics in detail, there was a great diversity of interests, at least within the dozen or so letters I have received so far. The folks that would call themselves the hackers (I emphasize that is their term), seem to need a "hacker's column". There are about three that responded that could do a much better job on hacker topics than I, and I suggest Don, that you get one of them to do a monthly bit for '68'. (added)

Editor's Note: I do not have the names of those who believe they could do a 'better' job of it. However, I would sure like to hear from some of them and look over some of their material. I guess it is possible that someone could do it better - but I wonder if **ANY** could do it longer as well? I wonder?

DMW

- - -

If any of us (readers of '68') are just software users, none responded. I suspect that all the software consumers or most of them at least, have switched to the other brands for which there is a large supply of "plug it in and use it" software. That being the case, there certainly is an obvious need for more "technical" information here, though I would think there would be some interest in languages and their implementations for the 6809. One letter did say that the things on FLEX were appreciated, but that the reader would be happy if I would continue things just as they had been going.

I received one rather interesting letter from someone who took issue with me over my discussion of function keys vs control codes. Unfortunately, I have misplaced the letter in my pile somewhere, but I remember what it said, so I'll do a rough paraphrase. It said that the writer disagrees with my feeling that function keys should not be used. It went on to describe the nice things for which function keys may be used in a database environment, in which the computer operator is essentially filling in the blanks in the database entry mode. Function keys can be programmed to output a whole string of characters and can be most useful in such applications. The letter went on to say that of course in text editing environments, the writer agreed that control keys are more useful.

About here I begin to wonder how carefully people read what I have written. First of all, I'd like to point out that the heading for the discussion in question was "Editors". Clearly I was referring to text editing applications in the major part of my discussion. I quote that column near the end of the discussion.

"I'm not saying that function keys are useless in something like a CAD system or perhaps a general ledger package in which each one will bring up a different menu or function, just that someone typing text ought to be able to control the cursor without looking at the keyboard." While I didn't mention Database entry applications specifically, the association should be fairly obvious. I don't think I disagreed with the writer of the letter at all, or at least only in mild degree. He did say that he realized that the columnist has the last word in such discussions, since he writes a column for a publication in a different field.

I still would like more feedback. The response was much as I anticipated and said, much to the annoyance of at least one reader. Out of the some 7000 (added: actually much more Ron, something like 15,000+, including newstand readers, overseas (57% of distribution), etc.) copies of '68' that are distributed, 12 people took the time to respond; not a very large percentage of the readers! (Don, how many responses do you get to your reader surveys?)

(added: Ron, if I include prepaid postage, or some other inducement, etc., response can run as high as 90% or better, for surveys of less than 200. One that was about 1880, was 100% - we called each one. For instance I gave a bad review of a book, "Fire in the Valley". I received about 100 or so letters about that particular subject. Over 90% agreed, and most were qualified. Fact is actually none disagreed, but a few thought I was too harsh. Then I went on a bit - in the June '85 issue - about our beginnings and some other nostalgia stuff. I have received 200-300 responses to that. I guess I'm just lucky, or something. However, for every letter you actually receive, you can figure at least a couple of hundred that agreed but did not reply.)

One subject I have received an extra large amount of feedback on is Tandy and the CoCo. I have been at odds with Tandy and some other magazines concerning the future of the CoCo, as we know it. Most was from small 'cottage' types who were selling hardware or software, or both, to that market. Although some were rather large concerns. The replies are heavy each time I comment, and practically 100% agree that things are falling pretty well as we called it. That is a declining market with no 'real' help in sight.

I get a lot of feedback from some of our present and past advertisers, with one central theme - 'Hindsight is always 20/20'! Especially when I talk about the difffculty and expense required to crack a Big Blue or Fruit market. So you see Ron, it is sorta like the old saying - "It's according to who's ox is being gored"! I think you under estimate the extent of your input. The engineer is always listening for that 'one' squeaky wheel, among hundreds. Also if he doesn't blow his whistle, who will?

DMW

- - -

At any rate, I really DO appreciate those of you who did respond. You have given me at least a LITTLE direction. I'll do my best to include more on FLEX. Maybe part of the problem is that I think the title of the column is flex USER notes, and some of you think it is FLEX user notes. I'll try to be more diverse and include something for everybody.

One thing became obvious as I read the letters that I've received so far. I got pretty strong with my bit about more CAT and COPY utilities, but I didn't make myself clear. Several readers took my words as meaning that I didn't want to see ANY utilities published in '68'. NOT SO, and I am sorry I wasn't more clear in what I was trying to say. Obviously, as some of the readers observed, if '68' Micro Journal didn't publish utilities there wouldn't be much reason left to continue publishing it. Those of you who have followed this column for a long time know that I've included utilities and other programs and fragments of programs right along. My comments were intended to be limited to CAT and COPY utilities and maybe one or two others (if the shoe fits). The point was that these two particular utilities have been programmed to death! TSC supplied CAT and COPY with FLEX from day 1. When they released their Utilities Package for sale, it contained FILES.CMD and DIR.CMD, both variations of CAT. FILES does a CAT in multiple columns as opposed to the original CAT which listed the contents of the disk in one narrow column (because SWTPc originally sold a 40 column printer) with as much information on each as the original CAT had. DIR is more comprehensive and lists starting and ending sectors, number of sectors, date, and kind of file protection for each file. I don't think I would be exaggerating if I said that at least 5 or 6 more CAT and COPY utilities have been published in '68' over the past few years. Enough said.

The one reader who really got nasty thought I was talking down to you readers. If any of you also felt that way, my apologies. The most recent letter was rather indignant about my suggestion that you could READ available books on Assembler programming and the Advanced Programmer's guide and absorb the information. "Why do you think there are schools and teachers?", he said. Another reader essentially said that I should write about details of FLEX in some depth. He had "learned by reading and re-reading articles in the early computer magazines until they made sense."

My outlook on learning from reading vs school is more in line with the comments of the second of those two readers. I graduated from engineering school in 1956 and I have been learning ever since, though I have not gone back to school. (In truth, I started a course once but had to quit because I got too busy at work.) My last electronics courses covered "Non-linear Vacuum Tube Circuits", and there was a brief section in a Physics course regarding solid state phenomena (transistors and diodes). My first job out of school I found that I would have to learn all about how to design circuits with transistors in a hurry. Off to the books. Then came digital logic RTL and DTL (with resistors, diodes and transistors at first). I had to dig and learn about Boolean Algebra and Karnaugh Maps for logic minimization, sequential logic, etc. Then came the integrated circuit revolution DTL, TTL, CMOS, the whole area of analog integrated operational amplifiers, A/D converters etc., and finally Microprocessors. Sorry, readers, but I never took a formal course on programming in ANY language. Just hit the books again. Of course the reading wasn't all it took. I had to try out what I learned. Do some logic design... simplify combinational logic via Karnaugh maPs... build some TTL circuits and make them work... start programming in machine code, assembler, BASIC, Pascal, Forth, C, PL/9, and a few others. I just don't buy the "You can't learn by reading." argument.

FLEX Files

Having so said, I'll continue the tutorial on FLEX. The reader who said one can't learn by reading indicated that file handling in FLEX is a problem. He has been able to copy someone else's program but not to write his own. Let's see if we can come up with a lucid explanation.

Flex (and just about any other operating system) uses a buffer of some sort for each file that is open. (A buffer is an area of memory used to hold data. FLEX calls this buffer area a File Control Block or FCB for short. The FLEX2 or FLEX9 FCB is 320 bytes (memory locations) long. The first 64 bytes contain information and workspace concerning the file, and the last 256 are used to hold one sector's worth of data. You could call those 256 bytes a "sector buffer". When a file is read, FLEX reads a whole sector at a time into the sector buffer portion of the FCB for that file, though the user access to the file is on a character at a time basis. FLEX returns a character from the buffer and increments a pointer to the buffer so next time you ask it for a character, you get the next character. When the buffer pointer reaches the end of the buffer, FLEX automatically reads the next sector and moves the pointer to the start of the buffer again. That is getting a little ahead of our story, however.

Let's do (of all things) a simple COPY program since that involves opening two files, reading from one and writing to the second. Refer to the accompanying listing for this explanation. If you want to talk files on the assembler program level, there are a few handy routines built into FLEX that make life a little easier for you. Essentially you reserve an area of memory for the FCB, point the X register at it, and jump to the FLEX routine GETFIL. GETFIL reads the filename information on the command line and places it properly in the FCB.

Suppose our program is called DUPLICAT.CMD. You type DUPLICAT FILE1 FILE2. FLEX loads DUPLICAT.CMD and starts running it. You point X at the FCB and JSR GETIL. GETFIL reads the command line buffer and gets the name FILE1. It puts the <u>name</u> in bytes 4 through 8 of the FCB, and (perhaps here is where the reader had his trouble) puts nulls ($00) in bytes 9 through 11 of the FCB. Filenames can be up to 8 characters long. They must be placed in the FCB starting at byte 4 (first byte of an FCB is byte 0) and extending through byte 11. If the filename is shorter than 8 characters, the remaining locations must be filled with $00. GETFIL will get the drive number specification from the command line if it is there and put it in byte 3 of the FCB. If no drive is specified, GETFIL puts the working drive number there. (The number is in HEX, $00 for drive 0 $01 for drive 1 etc. It is NOT in ASCII as $31 for drive 1).

If an extension was specified in the command line, GETFIL will place that in the next three locations after the filename, that is, bytes 12 to 14. If there was no extension none will be put there. Now, bring on SETEXT. FLEX SETEXT routine will supply a "default" extension that the programmer chooses when he writes the program. Let's use .TXT for a default extension. Look at Pg. 13 of the advanced programmer's guide. It tells me that in order to use SETEXT, I have to point X at the FCB (LDX #INFILE), load the A accumulator with the desired default extension code, in this case 1, (LDA #1 or LDA #$01 if you like). Now JSR SETEXT. If there was an extension specified in the command line, it will be left alone. If there was none, bytes 12 to 14 of the FCB will now contain "TXT".

NOW we are at last ready to open the file. Starting on page 29 of the programmer's guide, are the descriptions of the function codes. Function 1 is indicated as the Open for Read code. We put the $01 in the 0 byte of the INFILE by (LDA #$01, STA 0,X (since X is pointing at the start of the FCB anyway). Now we do a JSR FMS. FMS is a routine located at $D406. (Actually $D406 contains a jump to the actual FMS entry point). FMS tries to open the file and returns to the calling program. If FLEX was successful, the ZERO flag of the processor will be SET. If an error occurred, the ZERO flag will not be set and the code BNE ERROR will branch to your error handling routine.

Suppose for a moment that there was no such file to open for read. FMS would return with the processor ZERO flag cleared and the BNE ERROR would take the execution of the program to the error routine. There are a couple other useful routines in FLEX, again documented in the programmer's guide. One of them is the RPTERR routine. Your error handler should start out JSR RPTERR. The RPTERR routine will look at the second byte of the INFILE for an error code. If there is no such file, the error code is 4. RPTERR will go read ERRORS.SYS file if it is on the system disk and report FILE NOT FOUND. If it can't find ERRORS.SYS it will report DISK ERROR #4. The next two instructions in the Error handler should be JSR FMSCLS, which will close any open files, and then JMP WARMS, which gets you back to FLEX so you can try again with the program.

If the file is present on the disk and it is opened successfully, FMS returns with the ZERO flag set, and the BNE ERROR is passed by. One point that some of the software suppliers seem to have missed is that if the file is opened successfully, FMS changes the function code (1) that we put in the first byte of the FCB to a 0, which indicates that the file is open for read or write. That is, you don't have to explicitly put a 0 in the first byte of the FCB after opening the file. Now let's read the first byte of the file. If you've just called FMS to open the file, X will still be pointing at the FCB, but generally we read a file in a loop and do something else in the loop so that X might contain a different value. Therefore usually we point X at the FCB with LDX #INFILE again, and then just JSR FMS. If a byte is read from the file, again the ZERO flag is clear and the byte is in the A accumulator. If there was an error, a BNE ERROR will again get you to the error handler routine. After the last byte of a text file has been read, an attempt to read another will result in error #8. Usually the error routine checks to see if that was the error on en input file, and treets it as an end of file flag.

I am being a little general here but the assembler listing that accompanies this is full of comments that will help clarify each step in the process. Opening a file for WRITE is about the same process except that the function code is 2 rather than 1, for the Open for Write. If the file being opened already exists, FMS will return error #3. If the file does not already exist, it will be opened for WRITE. All the process of GETFIL and SETEXT are carried out the same way as for opening the input file. Of course you have to have specified two different file control blocks, the names of which are perfectly arbitrary. I like to use INFILE and OUTFIL or INFCB and OUTFCB to distinguish the input and output files more simply.

Back to our DUPLICAT program for a moment. After both files are open, we simply LDX #INFILE, JSR FMS, and we have the next input character in ACCA. Now we LDX #OUTFIL and JSR FMS with the character to be written in ACCA. If you try running this program you will be annoyed at the action of your two disk drives. It will read one character at a time and write one at a time. The result is that it reads 252 characters (one sector) and puts those in the OUTFIL sector buffer. Then FLEX sees that the input buffer is exhausted and the output buffer is full, and it writes the output sector buffer to a sector of the output file, and then reads another input sector. A good copy program uses a large block of memory, reading in many sectors from the input file and transferring them from the sector buffer to a large memory buffer. It then switches modes and transfers characters from the large buffer to the OUTFIL sector buffer. Such a program can empty the input buffer fast enough so that the disk drive doesn't deselect the head between sector reads. It can fill the output file sector buffer fast enough so that the write takes place without head deselection. Our dumb DUPLICAT program will clack the head select solenoids back and forth if you are copying from one drive to the other, and run the head back and forth from track to track if you are copying on the same disk (to a file of a different name, of course). The point of the exercise is not to write a better copy utility, but to show how to open files, read from them, write to them, and close them in an orderly manner.

Speaking of closing them, after the end of file is detected, you simply put function code 4 in INFILE first byte, point X at INFILE and JSR FMS. You do exactly the same thing with OUTFIL. If both files close without error you JMP WARMS and you are all done. It might be worth mentioning here, that this program is the basis for doing what is called "filtering". Suppose you have a "foreign" text file (one from another operating system, not one in German). Suppose that you received it over a modem, and that it is a BASIC program that is nearly compatible with your BASIC interpreter, but that it has LF to terminate lines rather than CR as FLEX text files have. Simply copy the file to another file using the program listed here with one little addition. Between the reading and writing of each character, while you have it in the A accumulator, you simply do the little routine shown in the listing. Essentially. you compare ACCA with a LF and if it matches, you load ACCA with CR before writing the output file.

That should bring to mind all sorts of interesting things you could do. Add a pseudo random number to each letter to encode a message. Write a program to subtract the same pseudorandom sequence from the encoded message and you have unscrambled it. If sender and recipient use the same random number generator program and start with the same seed, this works like magic.

You can convert upper to lower case or vice versa, remove all control characters (if you read a control character, just don't write that character to the output file), etc. I hope this has been a little helpful and perhaps has given you some ideas for writing some useful UTILITIES.

As a last minute note a week after last working on the above, no further responses have arrived in my mail. Next month I have a few things to say about Editors again.

```
                            NAM     DUPLICAT
                            TTL     READ A FILE AND WRITE TO ANOTHER
                            OPT     PAG
                            PAG
                    *
                    *
                    * THIS PROGRAM DEMONSTRATES FLEX HANDLING OF
                    * TEXT FILES
                    *
                    * EQUATES FOR FLEX ROUTINES
                    *
               0406 FMS     EQU     $0406
               0403 FMSCLS  EQU     $0403
               CD20 GETFIL  EQU     $CD2D
               CD3F RPTERR  EQU     $CD3F
               CD33 SETEXT  EQU     $CD33
               CD03 WARMS   EQU     $CD03
                    *
0000                        ORG     0
                    *
                    * GET INPUT FILE NAME
                    *
0000 8E    0076     START   LDX     #INFILE   POINT X AT THE FILE CTRL BLOCK
0003 BD    CD20             JSR     GETFIL    GET FILENAM FROM COMMAND LINE TO FCB
0006 25    44               BCS     ERROR     ERROR IF RETURNS WITH CARRY FLAG SET
0008 86    01               LDA     #1        DEFAULT EXT .TXT
000A BD    CD33             JSR     SETEXT    SET EXTENSION IF ONE WAS NOT SUPPLIED
                    *
                    * GET OUTPUT FILE NAME
                    *
000D 8E    01B6             LDX     #OUTFIL   POINT AT OUTPUT FILE CTRL BLOCK
0010 BD    CD20             JSR     GETFIL    GET FILENAME FROM COMMAND LINE
0013 25    37               BCS     ERROR
0015 86    01               LDA     #1        DEFAULT EXTENSION .TXT AGAIN
0017 BD    CD33             JSR     SETEXT
                    *
                    * OPEN INPUT FILE FOR READ
                    *
001A 8E    0076             LDX     #INFILE   POINT AT THE FCB AS USUAL
001D 86    01               LDA     #1        FUNCTION CODE FOR "READ"
001F A7    84               STA     0,X       FIRST BYTE OF FCB
0021 BD    0406             JSR     FMS       OPEN FOR READ
0024 26    26               BNE     ERROR
                    *
                    * OPEN OUTPUT FILE FOR WRITE
                    *
0026 8E    01B6             LDX     #OUTFIL   POINT AT FCB AGAIN
0029 86    02               LDA     #2        FUNCTION CODE FOR WRITE
002B A7    84               STA     0,X       FIRST BYTE OF FCB
002D BD    D406             JSR     FMS       OPEN FOR WRITE
0030 26    1A               BNE     ERROR
                    *
                    * NOW READ CHAR FROM INPUT FILE
                    *
0032 8E    0076     RWLOOP  LDX     #INFILE   START OF READ WRITE LOOP
0035 BD    0406             JSR     FMS
0038 26    12               BNE     ERROR
                    *
                    * NOW YOU HAVE A CHARACTER FROM THE INPUT FILE
                    * IN ACCA.  HERE YOU CAN INSERT CODE TO
                    * CONVERT CHARACTERS, ETC.
                    *
                    * SAMPLE CODE WILL CONVERT LF TO CR
                    *
003A 84    7F               ANDA    #$7F      REMOVE PARITY BIT ALWAYS FOR AS II TEXT
003C 81    0A               CMPA    #$0A      LINEFEED
003E 26    02               BNE     CONV1     NOT A LINEFEED, SKIP THE CHANGE
0040 86    0C               LDAA    #$0C      GET A CR
                    *
                    *END OF CONVERSION CODE
                    *
0042 8E    01B6     CONV1   LDX     #OUTFIL   POINT AT OUTFILE FCB
0045 BD    0406             JSR     FMS       WRITE TO OUTPUT FILE
0048 26    02               BNE     ERROR
004A 20    E6               BRA     RWLOOP    GO AROUND AGAIN UNTIL ERROR IS DETECTED
                    *
                    * ERROR HANDLER
                    *
                    * PROGRAM MUST EXIT VIA ERROR SINCE END OF FILE
                    * IS DETECTED BY PRESENCE OF ERROR CODE
                    *
004C A6    01       ERROR   LDA     1,X       GET ERROR CODE
004E 81    08               CMPA    #8        IS IT END OF FILE?
0050 27    09               BEQ     EXIT      IF YES, PROGRAM COMPLETED SUCCESSFULLY
0052 BD    CD3F             J R     RPTERR    ELSE REPORT ERROR AND RETURN TO FLEX
0055 BD    D403             JSR     FMSCLS    QUICK CLOSE OF ALL OPEN FILES
0058 7E    CD03             JMP     WARMS     ERROR EXIT
                    *
                    * NORMAL EXIT WITH CLOSE OF OPEN FILES
                    *
005B 8E    0076     EXIT    LDX     #INFILE   POINT AT THE FCB
005E 86    04               LDA     #4        CODE FOR CLOSE FILE
0060 A7    84               STA     0,X
0062 BD    0406             JSR     FMS       NORMAL CLOSE OF INFILE
0065 26    E5               BNE     ERROR     YOU CAN HAVE AN ERROR CLOSING A FILE TOO
0067 8E    01B6             LDX     #OUTFIL   NOW CLOSE THE OUTPUT FILE
006A 86    04               LDA     #4        CODE FOR CLOSE FILE
006C A7    84               STA     0,X
006E BD    D406             JSR     FMS       NORMAL CLOSE OF OUTFIL
0071 26    D9               BNE     ERROR
0073 7E    CD03             JMP     WARMS     ALL DONE, BACK TO FLEX
                    *
                    * FILE CONTROL BLOCK ALLOCATION
                    * IMMEDIATELY FOLLOWING PROGRAM CODE
                    *
0076                INFILE  RMB     320       INPUT FILE CONTROL BLOCK
01B6                OUTFIL  RMB     320       OUTPUT FILE CONTROL BLOCK
                    *
                            END     START

0 ERROR(S) DETECTED

SYMBOL TABLE:

CONV1  0042   ERROR  004C   EXIT   005B   FMS    0406   FMSCLS 0403
GETFIL CD20   INFILE 0076   OUTFIL 01B6   RPTERR CD3F   RWLOOP 0032
SETEXT CD33   START  0000   WARMS  CD03
```

OS-9 User Notes

Peter Dibble
19 Fountain Street
Rochester, NY 14620

Replay

Last month's column was a bit too quick of a job. I was very distracted by the end of the semester. I've decided to make this month's column a replay of last month's. As they say: keep doing it until you get it right!

School

The school year is over now. Even the qualifying exams are done. The department seems empty and (compared to the past month) calm.

I still recommend graduate school, at least the U of R Computer Science Department. Graduate students are at the bottom of the heap, but we have FUN! There's so tremendously much there is no time to learn when you live a normal life. I'm getting a chance to chip away at it. Of course, that just uncovers more...

I thought I was unusual in dropping a career to return to school, but a good fraction of the first-year class did the same thing. It's nice to know that I'm not the only one who gave in to the lure of graduate school after years of struggling against it.

If you don't find computers a stiff challenge any more, the conventional wisdom is to think about management. My suggestion: consider a graduate degree.

The Level Two Module Directory

Last month we left Microware with a problem. Where should they keep idle modules? Here's how they solved the problem:

The chunk of memory in the system address space dedicated to the module directory is filled with two different types of records. Module directory entries are in a table that grows stackwise (last-in/first-out) from one end. Each module directory entry includes a pointer to a DAT (Dynamic Address Translator) image. These DAT images are also stored in the module directory's block of memory in a table growing stackwise from the other end of the area.

A DAT image defines something that I like to call an address space. When the DAT image is loaded into the DAT hardware it defines the mapping of addresses in the processor to locations in the system's memory. The DAT images used for modules are unusual in that they are never loaded into the DAT. They only used by the module handling OS-9 services, never by the process scheduling services.

Several modules might share the same DAT image. If modules are loaded from the same file they all go into the same address space (with the same DAT image). To fully identify the location of a module in memory OS-9 uses the DAT image and the address (offset) in the address space defined by that DAT image. The length of the module is stored in the module header.

Module directory entries and module DAT images are allocated stackwise. Unfortunately for the OS-9 designers, they aren't freed the same way. We can be pleased that they didn't decide to force us to arrange our module usage around a stack. Imagine, you load a program (say copy) then you load another (say rename). You copy a few files than you want to free the memory up. You try to unlink the copy module, but you can't. First you must unlink rename, THEN you can unlink copy. If you still waot to do some more renaming you'd have to load rename again. Particularly when loading/linking/unlinking can be done implicitly, this stackwise bit would be no fun!

The solution for this problem is something between cute trick and kludge. It relies on a lucky break. Nothing ever remembers the address of a module directory entry or a module DAT image long enough to span the recognition, by F$VMOO, of a module. Whenever the address of a module directory entry is needed an appropriate system call is used to scan the directory and find the entry, then the address is used promptly and discarded

Since pointers into the module directory are never saved the directory can be "garbage collected" and compacted.

Deallocation of module directory entries can be done by flagging them as idle. This can leave empty slots in the middle of the module directory, but that's ok. The system knows the directory needs compaction when the module directory entry table and the OAT image table bump into each other. (This can only happen when a new module is being added to the directory.)

When the module directory and the module DAT image space meet, OS-9 compacts the structure. It goes through the directory entries squeezing out the idle space. This is pretty straightforward. Squeezing the DAT image area is trickier because there are pointers to the DAT images in the module directory entries, and because the DAT images vary in size depending on the size of the address space they define. Each time a module DAT image is moved the module directory entries must be searched for references to that DAT image. Each pointer to the DAT image must be adjusted to point to its new location.

If there is still no room for the new module directory entry after compaction, OS-9 will return an error 206 (Module directory full).

Garbage collection and compaction sounds pretty arduous. It is. But it's not a big issue. For one thing, if the module directory isn't filled almost to the brim it will only be compacted occasionally.

Another reason not to worry about the time spent in compaction is that it isn't that long.

The amount of time it takes to recover an idle entry depends on the number of entries between it and the active end of the directory (Remember the directory is added to stackwise. New entries are always added at one end). The worst-case time to compact the directory will depend on the size of the directory squared. The time for compacting the module DAT images is also order of the size squared.

My module directory contains 62 modules now (output from MDIR). If there were 61 idle entries scattered through the directory the time to do compaction would be about fifteen-million times the time to shift one directory entry. Assuming that it takes about 50 machine cycles to shift an entry one place we get about six minutes to do the compression.

Obviously it doesn't take that long. The important difference between reality and my aoalysis is that I included far too many idle module directory entries. In my system almost all the modules are loaded at system startup and never unlinked. Only the last five or ten modules in the directory are dynamic, so idle entries far from the active end of the table are rare. Guessing that after the top ten entries there are no idle slots in the directory we get:

(10^2)*50 cycles

or about three thousandths of a second for a compaction.

If you don't pack your module directory up to the limit, and you load/unlink modules in a generally last-in/first-out way; you may never see a compaction that has to shift more than one entry.

I think the choice of the module directory's data structure is interesting. I didn't think much of it when I first figured it out. There are other ways to arrange it that would not have needed compaction. When I looked harder it looked better. All the clean ways of organizing the module directory would have involved some form of linked list. Linked lists are fine, but it's quicker to search a table than a linked list. My guess is that someone at Microware decided to accept the kludginess of garbage collection and compaction because the result will almost always be faster.

There are a few practical results of the module directory's structure. One is that loading modules in a multi-user system (They fragment the module directory more) that has almost filled the module directory might get slow. Time spent doing compaction could also explain small changes in execution time.

Remember that module directory entries and module DAT images can move. Don't ever save a pointer into that area; it may become meaningless.

Notice that you can't find modules using only the module directory. You need the DAT images too. As I see it the only information in a module directory entry that is useful without the DAT image is the usage count for the module. I wonder what the F$GModDr system call is for.

It would be useful to know how big your module directory is. The limit on the size is 2048 bytes; that's the maximum that the F$GModDr SVC will return. I think most systems use 1536 bytes for the module directory. Check the D.ModDir and D.ModDir+2 fields in your system direct page to get a firm answer.

What follows is part of the module directory from my system:

```
Addr   0 1  2 3  4 5  6 7  8 9  A B  C D  E F
----  ---- ---- ---- ---- ---- ---- ---- ----
0A00  0FF8 0F06 0000 0001 0FF8 0F06 02B0 0000
0A10  0EA6 5F4E 0C00 0001 0EA6 5F4E 0C2E 0001
0A20  0EA6 5F4E 1892 0001 0EA6 5F4E 1907 0001
0A30  0EA6 5F4E 2230 0003 0EA6 5F4E 3305 0000
0A40  0EA6 5F4E 38FD 0000 0EA6 5F4E 3DBF 0000
0A50  0EA6 5F4E 4060 0000 0EA6 5F4E 42BE 0001
0A60  0EA6 5F4E 4407 0001 0EA6 5F4E 44FF 0001
0A70  0EA6 5F4E 4525 0001 0EA6 5F4E 4631 0003
0A80  0EA6 5F4E 49CD 0003 0EA6 5F4E 4DF2 0000
0A90  0EA6 5F4E 4E1E 0000 0EA6 5F4E 4E4A 0000
```

The module DAT images start at $1000 and grow downwards. Each directory entry is eight bytes long and contains four 2-byte fields. For the first entry those fields are:

Module DAT image pointer:	$0FF8
Address space size:	$0FD6
Offset to module:	$0000
Module Link Count:	$0001

Virtual Memory

My column in the July issue included comments on virtual memory. Now the details are out on the Gimix 68020 system. I just don't know how to react. A megabyte is a lot of memory; our IBM PC friends get a lot done with 640K. For the kind of thing that people do with personal computers these days a megabyte should be plenty. Even without virtual memory the Gimix 68020 system will be able to compile programs, do text editing, and handle any reasonable-sized spread sheet; it will do what I do on my home computer now -- much faster.

I get nervous when Richard Don invokes VIRTUAL MEMORY and starts talking about Franz Lisp, Prolog, and an Ada Compiler. I fear that someone has "sold him a bill of goods." I'm willing to be convinced. If anyone sees a real application running Franz Lisp on the Gimix please tell me about it. For real lisp applications see your nearest Artificial Intelligence fanatic. Suggest Macsyma (does calculus), Conniver (makes plans), or Argot (understands some English) to get him on the right track.

Users Group

I am an example of the Users Group losing people. Even though I am its Vice President somehow I slipped through the system when I moved. It's my fault I'm afraid. I bet I didn't send a change of address card in. Finally a collection of MODT newsletters caught up with me. I'm impressed.

The MOTD newsletter makes interesting reading. Greg Morse writes the "Basic09 Corner." In the latest issue he discusses subroutines and procedures. There's a column called "The COCO Advocate" by Jim Schmidt. In the last issue he talks about his experience with an MS-DOS machine from an OS-9 user's perspective. Lori Crovac wrote a rather horrifying article called "How to Roast a Computer." Let me quote her description of one of the ingredients:

1 - 4 Memory Boards Note: Both of these may also be found in the computer. Be sure to remove them from the computer carefully, and clean off those funny looking bits and pieces. I throw them in a pot in the freezer and use them later for soup.

The most interesting part of the latest MOTD, and a relief after reading that cooked ribbon cable tastes rather like spaghetti, was the Software Exchange News. Dave Kaleita and his group have made exceptional progress. Thirty-six disks are listed though some of them aren't ready yet.

Some highlights of the software library:

The Adventure game, both executable and Microware C source. The program is so big that the source fills a disk.

A set of useful C programming tools, mostly by Carl Kreider (A C beautifier, a program that finds function headers and lists them with their line numbers, a program that slices a Microware C object library into its component parts, a C cross reference generator, and simple formatted print program for C code).

XLisp. Read about it in the Byte Artificial Intelligence issue.

The User Notes Book

We have been pulling a trick that is commonly used on candy bars with my "User Notes" book. The price of printing went up so we shrunk it by thirty pages or so. The print is a little smaller and there's less empty space, but my critics say that it is still legible. I hope you like it.

Microware's New Quality Assurance Program

Microware is growing up. I admit that it is a good thing for all of us, but I don't like it. They used to be such a cozy little company. Now they've started a heavy duty quality assurance program. Somehow this reminds me of the Haynes underwear commercial. There are a group of three people at Microware responsible for catching and keeping track of bugs. Do you imagine inspector three saying "They don't say Microware until I say Microware!"

This new department at Microware should do a few nice things for us users. First, inspector 3 and his helpers should catch some of the bugs that we have been finding for them. Microware has always been pretty good about testing their stuff before they ship, but we can expect them to get better.

Bugs will slip through (Murphy's Law). In the interval before a new release with a different set of bugs comes out they plan to include a list of known bugs (with fixes and work arounds where they are known) in each software package. Nice idea. It used to be that we had to call the hot line to get fixes for known bugs. Now the free ninety-day support will be a little less important.

Big customers (OEMs and distributers) will get something like a newsletter with all the latest bugs and fixes. The information isn't secret, Microware just can't afford to send mailings out to every OS-9 user at frequent intervals. Check with the company you bought OS-9 from to find out their policy on this. Smoke sends out a newsletter don't they? Maybe they'll put highlights of the bug list in it.

Counting Blessings

I ran into some benchmark programs on Usenet a few weeks ago. I ran them on my machine. One was a complex set of C language programs. My machine (2 MHz 6809 running MW C) averaged about a quarter as fast as an eight Mhz 8086 running with no wait states. The 6809 looked best at procedure calls and worst at handling 32-bit integers. The benchmark programs were a sieve, a program that did floating point operations, a quicksort of long integers, and a program that calculated fibonachi numbers (integers). None of the benchmarks tested the 6809's strong points (as compared to 8088s), handling byte variables and interrupt service.

I didn't do anything to the programs. I imagine that I could have speeded some of the benchmarks up by using DIRECT variables.

The set of benchmark programs are too long to include here, but they were taken from the August 1983 Byte and modified to accommodate register variables. The Usenet note included statistics for several C compilers. The best all-around MS-DOS C compiler looked to me like the Microsoft C small model. The comparison between that and Microware C is:

		Microsoft	Microware
float	time	153.9	708
	size	18996	5834
sieve	time	2.91	9
	size	5844	3819
qsort	time	46.31	213
	size	10594	8810
fib	time	32.66	50
	size	5882	3814

The surprise was a very simple basic benchmark program:

```
5  TIME$ = "00:00:00"
10 for x=1 to 10000
20 if sqr(x) <> int(sqr(x)) then 40
30 print sqr(x),
40 next x
45 print TIME$
50 end
```

The execution times in the notes I read were:

IBM-PC	3 min. 19 sec
6MHz IBM-AT	1 min. 13 sec
5MHz Z-100	1 min. 13 sec
7.37MHz Z-100	49 sec
Sanyo (Sanyo Basic)	4 min. 37 sec
Sanyo (GW-Basic)	1 min. 40 sec

I converted the basic program to basic09:

```
PROCEDURE bnchbasic
PRINT DATE$
FOR x=1 TO 10000
     IF SQRT(x) = INT(SQRT(x)) THEN
          PRINT SQRT(x),
     ENDIF
NEXT x
PRINT DATE$
END
```

It ran in 1 min. 26 seconds. Compare that to an IBM-PC! From the C benchmarks it's clear that it isn't the raw power of the 6809 that makes the Basic09 benchmark faster than a PC. It must be Basic09 that is so fast.

Have you noticed that nobody sells copy protected software for OS-9. Every time I read Pournelle's column in Byte it makes me feel good about our software vendors.

"C" User Notes

Edgar M. (Bud) Pass, Ph.D.
1454 Latta Lane
Conyers, Ga 30207

INTRODUCTION

The C language, like many other algorithmic languages, allows the programmer great latitude in the formatting and organization of expressions, statements, functions, and programs. In contrast to some early versions of FORTRAN and BASIC, C allows multiple statements per line, does not require every statement to be labelled, supports a useful (although limited) block structure, allows meaningful variable names, implements structured and compound statements, and can be generally very supportive of good programming practices.

Unfortunately, the very freedom which allows the user so few restrictions also may lead to very bad programming practices, which detract from the usefulness of the programs. This situation is similar to that of a featureless geographic plain, which becomes more useful and less dangerous to motorized traffic after it is structured by roads, stop signs, traffic lights, road signs, etc.

GUIDELINES

In order to enhance the readability, reliability, usability, portability, and maintainability of C programs, certain guidelines are presented here. These guidelines are suggestive, rather than restrictive, and are provided to stimulate readers to develop programming guidelines of their own. These personal guidelines may become so natural that the programmer will develop a style of writing C programs which will be automatic and effortless. Many programmers who have developed such personal styles will edit others's programs into their own style, or write programs to automatically do so. Programs are also available for UNIX and BDS versions of C which provide such stylistic services, although many users further modify such programs to suit their styles.

The guidelines presented here range from detailed suggestions for the formatting of comments, statements, and expressions to general suggestions for the structuring of programs.

PREPROCESSOR

Program and machine constants, as well as implementation dependent declarations, should never be stated directly, but should be stated indirectly thru "#define" statements. This not only enhances the self-documenting aspects of the C program, but aids in program portability and eases maintenance difficulties.

Fields of "#defines" and "#includes" should be aligned. The defined identifier should be composed of all capital letters and digits.

For example, use

```
     #define    DEF1        134
     #define    MAXDEFN     3
```

and not

```
     #define DEF1 134
     #define MAXDEFN 3
```

Conditional compilation, using "#ifdef" and "#endif", should be used to control all special debugging and instrumentation statements and should be used in coordination with "#define", "#include", and "typedef" to control any code which may vary among versions of C compilers. When converting programs to any of the versions of C which do not support "#ifdef", the "#ifdef" and "#endif" directives may be commented out to include the code between them or the entire group may be commented out (being careful of nested comments) to suppress the code between them.

If a C program is composed of multiple modules which are separately compiled, place the global and external declarations and "#define"s into a common "#include" file, and precede each declaration with the pseudo-declaration "extern". Then, in the module which contains the "main" function, provide the following "#define" preceding the inclusion of the global declarations:

```
     #define ext
```

and in each of the other modules, provide the following "#define" preceding the inclusion of the global declarations:

```
     #define ext    extern
```

Parametric "#define"s should be minimized or avoided if any of the potential C compilers on which the program is to be run do not support parametric "#define"s. Many fewer implementations of C support parametric "#define"s than support simple "#define"s. Almost no Small C implementations support the parametric form.

DECLARATIONS

Global variables may begin with an upper case letter and otherwise be composed of lower case letters and digits. Local variables and function names should be composed of lower case letters and digits. Since some versions of C do not distinguish between upper and lower case in variables, function names, and labels, data names differing only in the case of their constituent letters should always be avoided. Also, since many versions of C allow only letters and digits in identifiers, the use of special characters, such as underlines, should be avoided, as reducing portability. Also, since some versions of C place limitations on the length of the number of significant characters in an identifier (usually 6, 8, or 31 if a limitation is imposed), it is advantageous to avoid long names which vary only near the end.

Clarity should override capitalization rules, as long as ambiguous situations as just described are not created; thus "PayDate" is preferable to "Paydate" for the name of a global variable, and either name is preferable to "P" or "Qwert". However, single letter variable names are acceptable for loop control and temporary computation purposes.

When the storage structure or type of a variable is important, always state it explicitly. For example, specify "auto" if the address of a local variable is to be found using the "&" operator (so that another programmer will not change its declaration to "register"). Also declare external typed functions explicitly so that unwanted type conversion to type "int" (or lack thereof) does not occur.

The order of declarations in a program or separately compiled program module is normally defined to be the local "#define"s, the global "#define"s, the declarative "#include"s, the "main" function, and other functions, usually in some logical order. In large programs, functions may be placed into alphabetical order, for lack of a better scheme, to assist in locating them.

COMMENTS

A fairly large comment block should exist at the beginning of each program or "#include"d program set, major function, and function subsection. This block should clarify such points as why the functions or declarations have been grouped together and any other information common to the functions or declaraions.

It is helpful to develop a template which may be copied into place, as required, to standardize the major comment format, and thus to prompt the programmer to enter the information. The first lines of the comment should contain the name of the module and a quick description. This should be followed by additional information including the parameters of the routines, any options that the user may specify, global variables used and modified, other functions called, date and purpose of each revision, etc. Comments defining the parameters, return codes, and side effects should be stated as required. Comment parameters and globals as being for input only, output only, or modified within the module. Blank lines should separate major points in the comments.

Block comments, as any other multi-line comments, should have the initial "/*" and final "*/" on separate lines from the text of the comment. Optionally, a common identification, such as two asterisks leading each line or forming a box surrounding the comment, may be consistently used to illuminate the block. Block comments explaining program functions, major functions, or "#include"d code should be left-justified. Other comments will generally appear at the same indentation level as the code they are explaining.

Other comments should be used liberally, and should explain, not mirror, the code being described. Very short comments may appear on the same line with the code, but most comments should appear on their own lines.

A blank line (or a null comment) should be used to separate the declarations and the statements on a function or block. Blank lines should also be used freely to separate minor subsections of code.

Remember, when you are commenting a program you are writing, that you may be reading your own comments six months after you have put the program aside. By then, you will be almost unfamiliar with the program and will be reading them as if you had not written it. So write the comments as you would like them to be written by someone else for you.

An example of a good block comment format, as used by the INGRES project at the University of California at Berkley, appears at the end of this discussion, with a sample program.

SYNTAX

It is easy to write totally incomprehensible code in C, but that is not usually the goal of most C programmers. It is not usually more difficult to write clear and correct code, and the code is much more useful if it is easier to debug completely, works reliably, and is easier to maintain. Almost no useful programs are never modified during their normal lifetimes, and, in many cases, more effort is placed into the maintenance of a program than was expended during its original definition and programming.

The structured constructs in C, such as "for", "while", "do", etc., should be preferentially used in many contexts in which "if (expr) goto label" would have been used in BASIC or FORTRAN. Programs which over-use the "goto" statement are sometimes called "bowls of spaghetti" because of the tangled control logic flows. Such programs are often exceedingly difficult to debug and maintain, and are often easier to rewrite into more structured code than they are to check out or modify. The "goto" statement is much maligned, but, like most other statements, can be used to great advantage in some contexts or badly abused in others.

Semicolons should be followed by spaces. Commas may or may not be followed by spaces. Binary operators may be surrounded on both sides by spaces. Unary operators should be in direct contact with their arguments, except for "sizeof", which should be separated by a space from its argument. In many cases, fully parenthesizing expressions is advisable, especially since various versions of C assign different heirarchial precedence to the binary operators.

Implicit type conversions should be understood and carefully controlled. When subtle type conversions are required, explicit type conversions should be coded. The next version of the C compiler on the same machine or a different version on another machine may have a different interpretation of the type conversion than that provided by the original C compiler, creating problems which may be very difficult to locate and correct.

Two statements should not usually be placed onto the same line. The exceptions are statements such as "if" and a single, short statement such as "goto label" and "for" or "while" with a single, short statement such as "i++", or null statements, in any case. If statements are actually compound, they should be placed on separate lines and indented, usually four spaces. The matching "else" for an "if" statement should be placed at the same indentation level as the original "if" if the entire statement is not placed on the same line. Contiguous, mutually exclusive "if" statements, should be separated with "else" clauses.

```
For example, use
     if (expr)
         stmt;
or
     if (expr) stmt;
or
     if (expr)
         {
              stmt1;
              :
              :
              stmtn;
         }
or
     if (expr) stmt1; else stmt2;
or
     if (expr)
         stmt1;
     else
         stmt2;
or
     if (expr)
         {
              stmt1;
              :
              :
              stmtn;
         }
     else
         {
              stmt1;
              :
              :
              stmtn;
         }
but not
     if (expr)
     stmt;
or
     if (expr) stmt1;
     else stmt2;
or
     if (expr) stmt1;
     else
         stmt2;
or
     if (expr) {stmt1; .... stmtn;};
```

Braces should usually be placed on separate lines. However, the closing brace ending a "do" statement should usually be placed on the same line as the "while", and braces involved with initializers and "struct" are usually placed on the same line with their arguments. Braces may start at the same indentation level as the statement with which they are grouped or at one higher indentation level, and statements inside the braces should be placed at one higher level. Matching braces should always be placed at the same indentation level. This will make programs easier to read, debug, and maintain, because the scope of compound statements and blocks will be more obvious.

```
For example, use
     while (expr)
         {
              stmt1;
              :
              :
              stmtn;
         }
or
     while (expr)
     {
         stmt1;
         :
         :
         stmtn;
     }
or
     do
         {
             stmt1;
             :
             :
             stmtn;
         } while (expr);
but not
     while (expr)
         {
         stmt1;
         :
         :
         stmtn;
         }
or
     while (expr)
     {
     stmt1;
     :
     :
     stmtn;
     }
```

```
or
     while (expr) {
         stmt1;
         :
         :
         stmtn;
     }
or
     do  {
             stmt1;
             :
             :
             stmtn;
         } while (expr);
```

There should always be a space before or after a C keyword, such as "do", "else", "for", "if", "while", etc., but never between a function and the parenthesis preceding its arguments.

```
For example, use
     if (expr)
         func(0);
or
     if (expr) func(0);
but not
     if(expr)
         func (0);
or
     if(expr) func (0);
```

Every case of a "switch" statement (including "default") may be preceded by a blank line. The keyword "case" or "default" should have the same indentation as that of the "switch" statement plus two spaces. It should be followed by a space and the case constant. Multiple case labels on a single block a single block of code should be on the same seperate lines, but they should not be separated by blank lines. The "case" statements should be listed in increasing order of the constant, and the "default" statement should appear last in the "switch" statement.

The "switch" statement should generally be used in place of such chained "if" statements such as ones in which all (or almost all) expressions involve equality of the same sub-expressions with various constants.

```
For example, rather than
     if (expr1) stmt1;
     else
     if (expr2) stmt2;
     else
     if (expr3) stmt3;
     :
     :
     else
         stmtn;
attempt to use
     switch (expr)
     {
       case c1:
         stmt1;
         break;
       case c2:
         stmt2;
         break;
       case c3:
         stmt3;
         break;
          :
          :
       default:
         stmt3;
         break;
     }
```

whenever the situation allows.

The implementation of "register" variables is very inconsistent among versions of C. For portability reasons, variables should not be declared directly, but indirectly, thru the use of a "#define" such as the following:

```
     #define reg    register
```

which may be changed to the following when "register" declarations are unwanted or illegal:

```
     #define reg
```

For those C compilers which support "register" variables, they should be used whenever possible and convenient. However, there are restrictions on the use of "register" variables. The only types which may be "register"s are "int"s, "char"s, and pointers. There may only be a limited number (usually three) of register variables per function. A register variable has no address, so, if "r" is a register variable, "&r" is illegal.

GENERAL

The "main" function should explicitly call "exit(x)", with the default return code being zero. Each typed function should explicity return a value with "return(x)", and each untyped function should explicitly call "return", with no argument. Typed functions should never be used in an untyped context, and untyped functions should never be used in a typed context.

The declaration of a variable as local, global, or parametric should be analyzed. Normally, variables used within the context of one function only should be declared local to that function. This practice protects the variables from inadvertent modification by other funcions and conserves global variable storage, which is overhead shared among all functions in a C program. However, variables which are shared among several functions may be declared global to minimize the overhead required to pass their addresses among the functions. A case of a variable which may be coneidered for global declaration, even though it may used only in one function, would be a large table or structure which would otherwise require re-initialization on each call. Although such variables may be declared as "static", many implementations of C do not support "static" variables or handle them internally as if they were global variables, and the lack of portability would often overwhelm the advantages.

The "main" function should carefully edit its arguments and provide appropriate prompts in case its arguments are unacceptable. The level of editing and the verbosity and variety of the prompts depends upon the expected level of user of the program. A useful default standard prompt for a program always expecting arguments would be a "help" providing all of the argument and option meanings.

Whenever possible, functions should be kept relatively short. This is for several reasons. One is that, in large programs, common code should be placed into common functions, making the overall program shorter and easier to debug and maintain. Another reason is that shorter functions are easier to write, debug, and maintain than longer functions, because they then have fewer requirements on their inputs, processing, and outputs. Still another is that many C compilers simply cannot process very long functions because of internal table restrictions.

SUMMARY

This discussion has presented a suggested set of guidelines for the writing and structuring of C programs. It is intended to help the reader develop a personal style of writing C programs which will enhance the readability, usability, efficiency, portability, reliability, and maintainability of their C programs.

The following example program was based on a longer example provided by the INGRES project at the University of California at Berkeley. It illustrates many of the C style concepts presented above.

```
/****************************************************************
**
**   Name:
**       example
**
**   Purpose:
**       provide a sample program
**
**   Usage:
**       example [flags] argument
**
**   Positional Parameters:
**       argument -- this gets echoed to the standard output
**
**   Flags:
**       -n -- don't put a newline at the end.
**       -x -- don't do anything.
**       -b -- echo it with a bell character.
**
**   Return Codes:
**       0 -- successful
**       else -- failure
**
**   Defined Constants:
**       XEQ1 -- maximum number of simultaneous equations
**
**   Compilation Flags:
**       XTRACE -- enable trace information
**
**   Trace Flags:
**       5 -- general debug
**       6 -- reserved for future use
**
**   Compilation Instructions:
**       cc -n example.c
**
**   Notes:
**       this comment has nothing to do with the program below!
**
**   Deficiencies:
**       it should handle pseudo tty's
**
****************************************************************/


#define reg        register
#define ext        extern
#define XEQ1       5
#include           "stdio.h"
struct magic
{
        char *name;          /* name of symbol */
        int type;            /* type of symbol, defined in symbol.h */
        int value;           /* optional value.  This is actually
                             ** the value if it is type "integer",
                             ** a pointer to the value if it is a
                             ** string.
                             */
};


struct magic Stuff;
main(argc, argv)
        int argc;
        char *argv[];
{
        reg struct magic *r;
        reg int i;
        reg int j;
        int timebuf[2];
        auto int status;
        /*
        ** Note that in the declarations of argc and argv above, all
        ** parameters of any function should be declared, even if they
        ** are of type int (which is the default).
        */
        r = &Stuff;
        /* initialize random # generator */
        time(timebuf);
        srand(timebuf[1]);
        /* scan Stuff structure */
        for (i = 0; i < XEQ1; i++)
            {
#ifdef XTRACE
                if (tTf(5, 13))
                    printf("switch on type %d\n", r->reltype);
#endif
                switch (r->type)
                {
                  case 0:
                  case 1:
                  case 2:
                    /* end of query */
                    printf("bye\n");
                    break;
                  case 3:
                    /* initialize */
                    printf("hi\n");
                    break;
                  default:
                    /*
                    ** be sure to print plenty of info on an error;
                    ** "syserr("bad reltype");" would not have been
                    ** sufficient.  However, don't make syserr
                    ** calls too verbose; they take up space in the
                    ** object module, and it will probably be
                    ** necessary to look at the code anyway.
                    */
                    syserr("main: bad type %d", r->type);
                }
            }
        if (i == 5)
            {
                i++;
                j = 4;
            }

        /* resist the temptation to say "} else {" */
        else
            i--;
        /* plot the results */
        do
            {
                i = rand() & 017;
                plot(i);
            } while (j--);
        /* wait for child processes to complete */
        wait(&status);
        /* end of run, print termination message and exit */
        for (i = 0; i < 2; i++)
            printf("bye ");
        printf("\n");
}
/****************************************************************
**
**  PLOT -- Plot a Bar-Graph
**
**      Does a simple plot on a terminal -- one line's worth.
**
**      Parameters:
**          n (IN) -- number of asterisks to plot
**
**      Returns:
**          none
**
**      Side Effects:
**          none
**
**      Deficiencies:
**          Should allow scaling.
**
****************************************************************/
plot(n)
        int n;
{
        reg int i;
        for (i = n; i-- > 0;) printf("*");
        printf("\n");
}
```

ADA^R And The 68000

BY
THEODORE F. ELBERT
THE UNIVERSITY OF WEST FLORIDA
PENSACOLA, FLORIDA 32514

PART 5 - ADA'S DATA TYPES AND CONTROL STRUCTURES

Modern block structured languages, of which the Ada language is an example, are characterized by a feature known as data typing. Each data object, whether it be a scalar or a composite object, is of a particular type. A type specifies both a permissible set of values and a permissible set of operations, and the typing features of a language relate to the manner in which the language enforces type compatibility during compilation.

Ada is a strongly typed language. Ada's data types usually have an identifier associated with them, but there may be objects whose type has no name, in which case the object is said to be of an anonymous data type. The strong typing features of the Ada language manifest themselves in the fact that the type of an object is checked whenever any operation is applied to the object, to ensure that there is type compatibility between the object and the operation. Type checking is done primarily at compile time, but there are occasions in which it may occur at elaboration time.

The strong typing in Ada is much more pronounced than in other languages such as Pascal because, for reasons to be explored later, nearly all objects in a properly defined Ada program are of a user-defined data type of some kind. Furthermore, two objects that can assume the same values--for example, integer values--can be of distinctly different types and therefore cannot be mixed in any kind of arithmetic operation. To the experienced programmer encountering the Ada language for the first time this feature may seem rather outrageous, but some reflection should indicate that the reliability and modifiability of a program is enhanced by strong typing. Typing conflicts often indicate some underlying problem in the program design, and these conflicts are detected at compile time in an Ada program. Languages that are not so strongly typed, or that provide implicit type conversion when typing conflicts are detected, produce less reliable software for this reason.

While a given Ada program may have many different data types declared within the program, these types all fall into one of four general classes:

- scalar data types
- composite data types
- access data types
- private data types.

It is important to note these are classes of data types. There is no Ada data type named SCALAR, for example, but there may be declared in an Ada program any number of named data types of this class, which are generically called scalar data types.

Scalar data types. The class of scalar data types has three sub-classes:

- integer data types
- real data types
- enumeration data types.

As with the primary classes, there are no data types named REAL or ENUMERATION, although there is a predefined INTEGER type. Rather, a program may declare named types belonging to one of these classes. Two different types belonging to the same class, for example, to the integer class, have the same properties but are distinctly different classes. Two integer data types have common arithmetic operations defined, but objects of these types cannot be mixed in an arithmetic operation.

Integer data types have associated with them a set of values which is a consecutive range of integers. The Ada language provides flexibility in the declaration of an integer type in that a range of values must be specified, and values outside the range are not permitted to be assigned to objects of that type.

Real data types in any programming language represent only approximations to the real numbers, since a finite number of bits must be used to represent the mantissa of a real data type. For example, if the mantissa--expressed as a fractional part always less than one--is represented by twenty-four bits, then there are only 2^{24} different mantissa values that objects of the type can assume. Values that are not precisely equal to one of these representable values are, by necessity, approximated by one of the representable values. The result, of course, is that the representation of real numbers in any machine is actually discrete; thus, the representation belies the continuous character of the real numbers. Most programming languages ignore this fact, leaving the consideration of the accuracy with which real numbers are represented to the programmer. The Ada language, on the other hand, explicitly recognizes the fact that real numbers are represented by a range of discrete values. The language rules are specific about how real data types are represented by these discrete values, which are called model numbers. The net result is that a programmer may declare real data types with a minimum guaranteed error bound in the representation of real numbers, a feature which greatly enhances the portability of Ada software. Error bounds are defined in terms of the model numbers of the type.

Real data types are further divided into two categories:

- floating point types, which provide a relative representation error bound

- fixed point types, which provide an absolute representation error bound.

Floating point types are analogous to real data types in any other language. They are represented by a fractional mantissa and an exponent, thus making the representational error bound a function of the value. That is, the model numbers are very close together for small values, but they become farther apart as the value increases. Fixed point types, on the other hand, are represented by integer multiples of a basic increment, making the model numbers equally spaced over the entire range of values. This flexibility in the representation of real numbers, together with the explicit recognition of errors that may be inherent in the representation, provide the Ada programmer with a great deal of control over the numeric properties of his solution.

Enumeration data types in the Ada language are almost identical to those found in Pascal and other modern languages. The programmer, in effect, declares his own data type by explicitly listing the values that objects of the type may assume. The values are simply identifiers or character literals, and they are called enumeration literals. The purpose of enumeration data types is to increase the readability--and hence the understandability and modifiability--of the software. In languages such as FORTRAN, for example, the setting of a switch would be represented by two integer values, while in an Ada program an enumeration data type with the values ON and OFF can be declared and used to represent the switch position.

Composite data types. The class of composite data types in Ada contains two subclasses:

- arrays
- records.

There are no data types called ARRAY or RECORD in an Ada program, but rather a program may declare a named type belonging to one of these classes. Composite data types are those types that comprise combinations of components in some specific format. Array types and record types in the Ada language are very similar to their counterparts in other languages. In an array, all components must be of the same data type, while a record may have components of a differing type. In an array, the components are indexed, while in a record they are named. Components of records and arrays can be of virtually any data type, permitting complex data structures such as arrays of records, records with array components, records with record components, and so on. The features of--and operations on--composite data types in Ada are much richer than those found in virtually any other language.

Access data types. The access type in Ada is analogous to the pointer type in other languages such as Pascal. There is no data type named ACCESS, but rather a program may declare named types belonging to this class. Ada's access types permit the programmer to accommodate situations in which dynamic storage allocation--allocation made during execution as the need is perceived by the program--is required. For example, a number of records--unknown at compile time--may be required in an on-line system. The use of access types will permit record objects to be created--that is, storage to be allocated by the run-time support system--as the need arises in program execution.

In the example just cited, the record objects created at run-time are not access type objects; rather, they are of a record type. Objects of an access type designate other objects—in this case record objects—which are identified by no other means; that is the designated objects have no name. Objects designated by access type objects can also be deallocated, thus releasing the storage for further allocation by the run-time support system. In fact, such deallocation is often automatically effected under certain specified conditions in a process known as garbage collection.

Private data types. Ada's private data types may seem strange to experienced programmers encountering the Ada language for the first time. In the Ada language, a type defines a set of values and a set of operations for the type. The usual concept of a data type implies that the user is cognizant of the structure of the type and the set of values that objects of the type may assume. Private data types provide the programmer the ability to make useless to the user of a type any information regarding the structure and values of tbe type, and to limit operations on objects of the type to an explicit set of operations. This ability may not, at first consideration, seem to be of any particular advantage. However, the ability to declare private data types has one very important feature of permitting a programmer to produce abstract data types of his own design. An example may help clarify this statement.

In Pascal there is a data type named SET. In certain applications, this data type is very useful since objects of type SET have most of the characteristics of mathematical sets. Pascal objects of type SET have defined for them certain operations, such as union, intersection, and others that relate to manipulation of mathematical sets. The user is aware of these permissible operations on objects of type SET, but just how the objects are implemented by the language is hidden. It is not that the language implementer did not want the user to know how objects of type SET are implemented--this information could be obtained by examining the compiler--it is just that such knowledge is useless to the user because it could in no way affect how a program involving the use of the SET type is written. Type SET is an abstract data type--objects of the type can be declared and manipulated using a specified set of operations, but the implementation details are hidden from the user. Ada's private data types have precisely this feature, and therefore they serve to provide the programmer with the ability to create abstract data types with features specifically tailored to his particular application. For example, an abstract data type equivalent in every way to Pascal's SET type can be created in an Ada program by use of the private data type.

While declarations are the means by which an Ada program creates types and objects. statements are the means by which data is manipulated. While declarations are "elaborated", statements are "executed". Many of the Ada statements are conventional in nature--that is, they are similar to statements found in other modern languages. Specifically, Ada has the following statements:

- assignment   if
- exit   case
- goto   loop.

The control structures in Ada are similar to those found in any modern language. The **if** and **case** statements constitute the conditional control statements. The **case** statement has a slightly different syntax than that found in languages such as Pascal, but the basic function is the same. Ada's **if** statement has the usual if_then_else format, with nested **if**'s included by the reserved word **elsif**.

The iterative structure in Ada is the loop, which is bounded by the reserved words loop and **end** loop. Controlled looping is provided by either a **while** or a **for** condition. An Ada loop can be exited from any location by execution of an **exit** statement. Two other statements,

- procedure call   - return

are used in referencing subprograms. The return statement effects a return from a subprogram from a point other than the end statement of the subprogram body.

The remaining statements relate to the more advanced concepts of the Ada language. They are:

- delay	-- produces a time delay in the execution of an Ada task.
- raise	— used to raise a user-defined exception.
- entry call	-- used by an Ada task to request rendezvous with another task.
- accept	-- used by an Ada task to accept an entry call.
- select	-- provides flexibility in the use of entry call statements and accept statements.
- code	-- used to insert machine code mnemonic instructions into an Ada program.
- abort	-- provides a means of terminating tasks that do not terminate in a normal fashion.
- block	-- provides for the encapsulation of Ada statements, including local declarations, anywhere within an Ada program.

The Ada language permits a sequence of statements to appear anywhere a single statement may appear, a feature that enhances the maintainability of Ada code because it permits the insertion of statements without the insertion of the begin and end delimiters required in other languages such as Pascal. Instead, the Ada language uses the begin and end delimiters to designate sequences of statements that may have declarative regions--regions in which object and type declarations may appear--associated with them. The semicolon delimiter is used as a statement terminator; thus, every statement must end with a semicolon.

The Ada language does not contain a large number of different statements. Those statements provided by the language are somewhat flexible, and they tend to support modern program design methodologies.

NEXT: Ada's Subprograms.

[R] Ada is a trademark of the U.S. Department of Defense.

Basic OS-9

Ron Voigts
2024 Baldwin Ct.
Glendale Hts., IL 60137

OS-9 SMORGASBORD

This month is a smorgasbord. When I was younger, my parents and I would go to restaurants called "Smorgasbords". These establishments would specialize in serving buffets with a large variety of food. Instead of eating just one thing for dinner, it was more fun to try a little of everything. This month, instead of serving a big meal on some OS-9 matter, I thought I would serve a bunch of smaller ones. Actually these are a few things I have observed or come across in the past year of writing this column. They aren't large enough to develop an entire column around them and yet I think they are worth serving to my guests. So here is an "OS-9 Smorgasbord"!

A LITTLE MORE MEMORY

One of the big concerns for people running Level I, OS-9 is memory (or lack thereof). Usually the lack of memory doesn't become apparent until you run something big, like Basic09. You enter Basic09 by typing:

```
     OS9:BASIC09
```

Normally it gives you about 4k of memory to start, but you can request more. If you enter:

```
     B:MEM 20900
     20991
```

you get almost 21K of memory for the Basic09 workspace. The MEM command asks for memory and the amount given is returned. It is allocated in the nearest page increment or 256 bytes. That is why asking for 20900 gives you 20991. You can ask for more but this is the end of the line on my system. Let's start over. This time we can load Basic09 with:

```
     OS9:EX BASIC09
```

Now we can go for maximum memory, so:

```
     B:MEM 21700
     21759
```

This time we have 3/4K more memory. That's 3 pages more than before. This might not seem like much, but for a large program that extra bit might be very helpful.

So where did the extra 3 pages of memory come from? When we executed Basic09 the first time, the Shell went into a "wait" state after it started the process, Basic09. In the second example, The "EX" caused the shell to terminate itself after Basic09 started running. This left us with more available memory. There is a price to pay for this extra memory. When you leave Basic09, there will be no waiting shell! So you'll come back to a new shell. The execution and working directories will have to be redefined again.

This works not just for Basic09, but for other processes, as well. It is best to use it judiciously. Most things don't need the extra memory. But for something that needs more memory, this might help out.

WHERE DID THE MEMORY GO?

This has happened to me more than once and it can be perplexing if you don't realize what has happened. Usually I like to have my most used commands in memory when working with OS-9. This cuts down on the time waiting for the command to be loaded when you use it. One day I started Basic09 and then realized that I hadn't preloaded anything. I knew I would need the DIR command, so I entered:

```
     B:$LOAD DIR
```

The "$" causes Basic09 to pass the commands that follow to the OS-9 Shell for processing. LOAD put DIR into memory where it would be handier to use. Later I left Basic09 to do other things. I loaded the other commands I use often into memory with a:

```
     OS9:LOAD DEL LIST
```

Still later I wanted to go back to Basic09, so I typed:

```
     OS9:BASIC09
     Error #207
```

Now error #207 means Out of Memory. I knew DIR, DEL and LIST were loaded in memory, but I had used them before with Basic09 and no trouble. Where did all the memory go?

I was a victim of memory fragmentation. This occurs when you plunk a module in memory and leave open memory around it. I had loaded Basic09 before and then Dir. When I removed Basic09 later and added a few modules, my memory became split into two large chunks with Dir separating them. I ran Mfree and it reported:

```
     Address  pages
     -------- -----
      B00-54FF   74
     5800-AFFF   88
     B400-B4FF    1
     Total pages free = 163
     Graphics Memory at: $ 220
```

Dir was located at $5500 to $57FF. It divided memory into large chunks, but neither was large enough to hold Basic09.

Memory fragmentation occurs in Level I systems. Level II people are spared this problem, since their memory is dynamically managed. If you are a level I user and find yourself in this predicament, you'll have to remove something from memory. To remedy my solution, I removed everything I had preloaded with

```
    OS9:unlink del dir list
```

The UNLINK command removed these modules from memory. Then I started over from scratch.

ERRORS

One of cryptic parts of OS-9 is its error messages. Enter something disagreeable to the system and you'll get an error code number. You can use the command PRINTERR and every time an error occurs you'll receive the english translation of what went wrong. There are a few drawbacks to Printerr. First, it uses memory (memory, memory, always memory!) that you may not want to spare. Also, it does take time for it to look up the error message in /DO/SYS/errmsg. I understand people with Level II do not have "PRINTERR". (Editor's Note: Something about Microware "does not feel" that ALL Users on the System would want to see the errors printed out??? Personally, I would rather have the choice; or better still, it should be controllable just like "pause" or "echo" with the tmode command. Even so, what is the difference between a report of

```
    Error #216
```

and

```
    Error #216 - Path Name not found
```

except that the second one sure provides a lot more information. Each report takes up a line on the Terminal, and memory on a Level II System is not the problem it is with Level I. "Where's the Beef??" If it were not for the handy little "OS-9/6809 REFERENCE CARD" - available from Microware, with a LOT of good information on it including the Error Number descriptions, a summary of the OS-9 System Calls, File System I/O Paths and File Access Codes, Memory Module Header Format summary, and a 'quick reference' guide to the normal Commands... -- rlm)

An alternative to paging through your OS-9 user manual is to list to the printer /DO/SYS/errmsg which is the file that contains the error messages. Or you can photo copy the pages from your OS-9 manual that have a listing of the error messages. Take the list and post it in a prominent place near your computer. Next time you get an Error #216 or a Error #207, you can look at your list.

RADIO SHACK C LANGUAGE

I got a letter last month from a reader who had a Color Computer, Printer, and one Disk Drive. He had bought OS-9 and the C language compiler from Radio Shack. He wondered what else he needed to run C language. Well, the answer is another Disk Drive. The claim is that all you need to run OS-9 is a 64K Color Computer and a disk drive. This is true, but if you want to use the C language compiler, or do much of anything else, you had better plan on another drive for your system (and RS sure needs to support Double-Sided Drives for OS-9).

The C language that Radio Shacks sells is straight from Microware. It is a very formidable compiler. You must use two disk drives for compiling C programs. Drive 1 holds the disk with all the C compiler commands disk. On a 630 sector disk, this leaves 277 sectors for programs. That is why you need 2 disk drives to run C. in /DO/CMDS. There are 7 programs used for creating an executable module from the C source code, and most of them are long. For example, C.PASS1 is 125 sectors long and C.ASM is 82 sectors. Toss in the other 5 and you're looking at 488 sectors. And that's not counting regular OS-9 commands like DIR, LIST, and COPY. My C language disk for drive /DO has 610 sectors in use. The other drives disk has four directories -- DEFS, LIB, SOURCES and SYS. The files in these sectors are used in compiling the C program. They occupy 353 sectors on the

CAN YOU KEEP A SECRET?

Privacy of an OS-9 file can be very important. Perhaps you want it so only certain people can read your files. Maybe you want to leave messages on your system so only a particular user can read it. OS-9 lets you assign attributes to your files. You can make them a "read" file or a "public read". But how do you make them so only certain people can read them? The solution is to encode your programs and give the keys to unlock the code to whoever you want to read it.

This months program is a C language program called Crypt. It reads from the standard input path and writes to the standard output path. It expects one parameter, a keyword that is used to encode and decode the file. What makes the program work is the use of the "exclusive or", usually abbreviated XOR. XORing two 0's or two 1's yields a 0. XORing a 1 and 0 (or 0 and 1) is a 1. A byte can be XORed with another byte to give a new byte. If the new byte is XORed with one of the original bytes the result is the other byte. For example,

```
   $A6 XOR $57 = $F1
```

Now, we can XOR the $F1 with either $A6 or $57. Let's try $57.

```
   $F1 XOR $57 = $A6
```

The result is $A6. If we had used the $A6, the result would have been $57. So, what good is all this? By XORing a file with the bytes in the keyword, we create a new file of what would appear to be "garbage". To return the file to its original form, we XOR it once more with the keyword. The line that reads:

```
   c^=s[count++]
```

is XORing a character, c, with a byte in the keyword pointed at by s.

This C program is not too hard to understand. Argc should be equal to 2. Argv[0] points to the command, crypt, and argv[1] points to the key. The main body of the program is made of an if...else structure. If argc is not equal to 2, it prints a line that tells how to use crypt. Otherwise it inputs a character at a time and XOR's it with a character in the key. The pointer count is incremented and then it outputs the character. When the pointer count reaches the end of the keyword, pointing at '0', the key is used over again. To use it, enter like:

```
   crypt secretword <myfile >newfile
```

The word "secretword" becomes the key for the coding the file "myfile". "Newfile" will probably be unlistable and definitely makes no sense to anyone who doesn't know the key. You don't have to make files. For example, to read "newfile", you can try:

```
   crypt secretword <newfile >/p
```

This time crypt unscrambles "newfile" and lists it to the printer. Remember when using crypt, the key used to encode a file is also used to decode. So, use something you'll remember!

Well that concludes this month of OS-9 smorgasbord. If you have tidbits of information to share with the readers of this Column or want to pass on some helpful hint, drop me a note (include a SASE if you expect a reply). I'll pass on the information. I also will be sure to mention who sent it.

Bye for now!

68000 User Notes

Phillip Lucido
2320 Saratoga Drive
Sharpsville, Pa 16150

Revisiting the Time Program

There's a minor problem with the time program from this column two months ago. In the subroutine exec(), a line was omitted. This should be obvious to those typing the program in, since there is a gap of several blank lines where the line should be. The missing line, with the two lines around it, is

```
   if (child == -1)
      exit( errmsg(errno,
        "fork to `%s' failed - ",name));
   do {
```

More Mac Stuff

I've been on vacation for the past several weeks, so there aren't any new programs this month. I did take my Macintosh with me when I went to visit my family in St. Louis, but somehow I never managed to do anything other than play a Mac version of Asteroids. Seems that vacation, travel, and programming just couldn't mesh this time.

Still, I'm not entirely without topics. After all, I did just receive the **Inside Macintosh** technical manual a short time ago, and last month's introduction has barely scratched the surface. This month, I'll continue, going a little deeper into some aspects of programming on the Mac.

Traps Can Sometimes Be Useful

The Macintosh programming environment is based on a number of built-in routines. Most of these are found in the 64K of ROM, though some are actually in RAM, either because the ROM version has some bugs, or there isn't room in the ROM.

These routines are much like the OS-9 system calls, in that they use instructions which cause a 68000 exception through a vector in low memory. They do not use one of the TRAP instructions, though. Instead, each routine call is a single word-length instruction starting with $A. The 68000 has no legal instructions starting with $A, and executing such an instruction causes an exception, through the A-line exception vector.

The exception enters the trap dispatcher, which uses the remainder of the bits in the instruction to determine which routine is being called. Up to 512 routines are possible, with the bottom 9 bits of the trap instruction giving the routine number. This number is used as an index into a trap dispatch table, which holds the address of the entry point to the routine, either in ROM or RAM. By changing the entry within the trap dispatch table, it is possible to replace a routine in ROM with a new one in RAM.

Most trap routines require arguments, so some sort of calling convention from assembly language is required. Actually, there are two separate conventions. Some routines are stack-based, while others are register-based. Generally, high level (Toolbox) routines are stack-based and low level (Operating System) routines register-based.

Stack-based routines are meant to be called from high level language (HLL) programs. The format of parameters on the stack is that used by Apple's version of Pascal on the Lisa, where the Mac code was in large part developed. To call a stack-based routine, the following steps must be followed from assembly language. First, space is reserved on the stack for the size of the result, but only if one is to be returned. Next, the actual parameters are pushed onto the stack in forward order, as seen in the Pascal definition for the routine. Finally, the routine is called using the proper $A instruction. On return, the return address and parameters will have been released from the stack, with the optional result at the top of the stack.

This stack format, based on Pascal conventions, is incompatible with normal C conventions. In C, a return value is passed through a register, usually D0 or A0, and the routine parameters are pushed onto the stack in reverse order. The way in which various C compilers deal with this difference affects the size and speed of the code generated. The simplest fix, from the compiler writer's viewpoint, is simply to have the C statement actually call a short assembly language routine which converts the C stack format to the Pascal format. A nicer method, used in Aztec C and others, is to mark certain functions as being called with the Pascal conventions instead of C conventions, and generate the proper code in response. This generates the tightest possible code, but isn't strictly legitimate C syntactically.

Register-based routines are more like normal assembly language functions. Arguments are passed to the routine via registers, usually D0 and/or A0. In case of multiple parameters, A0 generally points to a parameter block in memory, rather than placing all of the arguments in registers. The register-based routines cannot be called directly from HLL programs, since these almost always use the stack for argument passing. To call these from C or Pascal, then, a small "glue" routine moves the arguments from the stack to the registers before calling the trap routine. Because of these "glue" routines, not all register-based routines are available with many compilers.

How to Organize Your Memory

As any impatient Mac owner knows, it is taking a long time for the forecast flood of software to make its appearance. One common excuse is the difficulty in writing Mac programs. If you read these excuses, one area in particular gets blamed most often. This is the Macintosh's method of managing RAM memory.

On most computers, the RAM memory is treated as one large, contiguous array of bytes, all of which is available to the running program. Even OS-9, with its multitasking environment, treats memory this way to some extent. The Macintosh has chosen a different scheme, involving a data structure called a heap, as well as the more normal stack.

In a heap, the memory array is controlled by the operating system, which allocates blocks of memory in response to requests from running programs. The programs have no control over the addresses of the allocated blocks. In addition, programs may release blocks which have been previously allocated, returning them to the pool of free blocks within the heap. Such a data structure is used by the standard C functions malloc() and free(). After many allocations and deallocations, the free memory within the heap may become badly fragmented, so the operating system may not be able to satisfy a request for a large block, even though the total size of all free blocks is sufficient.

The Macintosh uses the normal heap structure, but with some enhancements to deal with this fragmentation predicament. Most of the time, when a program requests a block from the heap, the block that is returned is made relocatable. If the Mac later discovers that no free block of sufficient size is available for a new request, it will move around the previously allocated blocks in an attempt to compact the memory, coalescing the free blocks to form one large free space.

There are two problems with this scheme. First, wheo a block is allocated to a program, the program must be told the address of the block. If the block is later moved, the address will have changed, and any pointer variables in the program holding this address will be left pointing to the wrong location. Since there may be a number of these pointers at unknowo locations in the program, the operating system cannot be expected to update them all when the block moves.

The method the Mac uses to get around this difficulty is quite elegant. When a relocatable block of memory is allocated by the operating system, a single pointer to the block, called the **master pointer**, is created. The operating system then returns a pointer to the master pointer to the program that made the allocation request. Tbis "pointer to a pointer to a block" is known as a **handle**. All references to the new block are made via double indirection through the handle, with the master pointer retaining the only copy of the actual address of the block. When the block must later be moved, only the master pointer needs to be updated.

There is a second problem with relocatable blocks. Some blocks, when allocated, may not be allowed to move. For instance, while the handle/master pointer allows blocks to be relocatable, the master pointer itself may not be moved. For special cases such as these, the Mac allows some blocks to be nonrelocatable. Since such blocks lead to heap fragmentation, the Mac will attempt to allocate them at the lowest possible position in the heap, perhaps moving relocatable blocks upwards in memory first.

There are many further refinements to the heap. First, there are actually two heaps. The system heap is reserved for the operating system, and consists of blocks used in low level I/O routines or other such things which must stick around all of the time. The application heap, on the other hand, is used to hold any program which is run, as well as all the data blocks used by the program. The system heap is kept through program invocations, while the application heap is wiped clean with each new program.

Relocatable blocks have additional complications. Sometimes, after compacting the heap, there still isn't enough memory available. The Mac will then look for blocks that have been marked "purgeable", and reclaim the space they use without waiting for explicit release trap calls by the allocating programs. Purgeable blocks generally hold data read from the disk (such as fonts) that can be reread if necessary at a later time. Of course, every time the program expects to use a purgeable block, it must make sure that its data is actually present in memory.

Finally, relocatable blocks may be marked as locked, so they will not be moved during heap compaction. This locked status is a flag which can be set or reset, so it is not the same as a nonrelocatable block. It is used when a block mustn't move for a short time only.

Obviously, with blocks of memory flying here and there through the heap, with visions of "hocus pocus, now you see it, now you don't," there is a great deal to go wrong in trying to use the heap. Unfortunately, it's the only game in town, and anybody tying to develop software for the Mac will have to thoroughly understand the concepts involved.

Use the Resources Available to You

I think I've room for one more topic this month. This is the concept of resources, somewhat analogous to the module idea found in OS-9. A Macintosh resource is a piece of code or data which is capsulized for easier use by a program.

A resource has a type, which identifies the type of data within it. For instance, all resources which declare the format of windows have resource type WIND, and code which defines a desk accessory or I/O driver is of type DRVR. Within a resource type, a resource has a unique ID number used to identify the particular resource. The resources are kept on disk in resource files, and are read into memory as blocks on the heap.

There are two main reasons for using resources in the Mac. First, there are certain objects which should be available to all programs. For instance, the various fonts, desk accessories, cursor patterns, and such do not need to be repeated within each program. Instead, they are collected in one system resource file, which is automatically searched when a program requests access to a resource. Second, abstracting the various windows, menus, and pictures used within a single program into independent modules eases development and modification of the program, by allowing just the needed changes to be implemented without worrying about the effects on the remainder of the program.

The Macintosh programming environment defines dozens of predefined resource types, with system traps to directly handle them. This encourages use of resources, rather than trying to perform their work using lower level system calls. In addition, a programmer is free to develop his or her own resource types, for use within a single program or between related programs.

CoCo User Notes

by Carl Mann
30 Warren Ave.
Amesbury, MA 01913

"COMPUPHOBIA"
or,
Who's Afraid of the Big, Bad CoCo?

Okay - I confess. There's no more enthusiastic supporter of a new thing than one who once hated it. That goes for just about everything, I daresay: being rich, stopping smoking for real and for good, jogging, sex preference, you name it. Show me someone who hates, fears, or despises a thing and I'll show you a prime candidate for Divine Revelation that will Change His (or Her) Life Forever.

So, I have to confess: in all good faith, I know exactly how many folks feel about the prospect of a computer entering a previously undisturbed lifestyle. I've been there myself. Boy, was it scary! First Serious Objection came out of my mouth at age nine or so: "I just don't want to be stuck in a room in front of a keyboard all day, pushing buttons!" Then there was the Great Grey Threat: "The Government will put us all in computers and make us do everything it wants us to do without asking first!" Last came the Appeal to Religion: "The Computer is the Tool of the Devil and all its users have the Mark of the Beast! The Beast will reveal itself as a Huge, Evil Computer!" That one was the scariest one of all, because my mortal soul seemed to hang in the balance. At the time, lots of other people around me were saying the same thing, so it appeared that it all must be true.

Odd, though...; most of those good and righteous folks were not only afraid of computers, but also of at least three of the other four things I first mentioned. There were other fears, of course, but those were the biggies. The most common exception seemed to be jogging, mostly. Nearly everybody in that bunch LOVED to run! (I wonder if that had any symbolic meaning?) I think I started to drift away from that crowd about the time I started asking questions like that. They didn't really seem to mind. I think I made them uncomfortable. Funny about that - I liked them, and still do.

Time passes, things change, and I forgot all about my fears. Don't get me wrong: they still existed. I just forgot about 'em as I fixed TV sets, took and passed the high-school equivalency exam, took a little college, opened a business, went bust, moved East, and took a job in a nuclear research instrument factory near the ocean. About five years into the job, I took interest in computers again.

Seems we had this small problem with our test records. The performance data we generated as we tested the only-one-on-the-whole-planet equipment we built was hand-written in a laboratory logbook. Once the system under test was shipped, the data was forgotten. Then six months to a year would pass while the system waited on the loading dock of some research lab and the paperwork oozed imperceptably from one department to another. Then came the day to actually install the contraption - but what did we set all those dials to the last time we got the darn thing to do what it barely did before??? Grab the logbooks, pull up a chair, thumb through all that scribbling, and maybe in twenty minutes we would have the data again. Maybe not. That got me to thinking.

The thought went something like this: "Suppose we could store all this information in a computer? Then we could just push a few buttons, turn on a printer, and get it all back in a flash! Wouldn't that be nice?" The boss agreed with unaccustomed vigor on two or three occasions. Finally, I up and did some research into database management software for the company's DEC PDP-11 - the one that nobody else seemed to be able to use.

I found what I was looking for, too. I don't remember the title of the program - only that it promised to do everything we needed a computer to do. I showed the reference listing to my boss. He shut the office door, jerked a thumb at the only other chair with a look at me that would have curdled a bowl of Wheaties, and leaned back in his own seat. "What makes you think I would let somebody like you play with that computer?" he growled. "If you think you want to play with computers, go buy your own! Now get back to work, and this conversation never happened!"

He was funny that way. Once he offered me money to go to school on - full time college - out of the allegedly depleted company treasury. I smelled a rat - he was famous for setting employees up for big falls - and declined with thanks. That conversation never happened either, and he'll confirm it. Wanna have some fun? I know this compulsive sadist that likes to play games...

I took old Pete's advice that payday. Instead of a new pair of shoes I had wanted, and a couple of sensual pleasures I had planned all week to indulge in, I marched right from the bank to the Radio Shack and bought a 32K Color Computer. I proceeded home across the parking lot with my prize and didn't wait to have dinner before plugging it in.

I got as far as connecting the little beastie to the TV set, turning said TV set on, and finding CoCo's power switch. Then it hit me. In about fifty nanoseconds, I realized that what I was doing violated every pronouncement I had ever promulgated concerning society, life, and living with (or against) computers! My brain did a long, slow barrel roll. Then it dived wild-eyed for cover as a blazing Technicolor vision (with full Dolby stereo) of the Devil a-rising through the slots on the left side of the machine's case (and his imps from the right-hand ones) engulfed my imagination. (I wonder - did John of Patmos serve a similar apprenticeship? Guess it comes down to who you want to trust...)

I shook free of the storm. After all, I had just mortgaged myself into next Tuesday for the thing, and the Devil clearly had no right to interfere with human progress. I pushed the button, and the now-familiar title, "EXTENDED COLOR BASIC 1.1 (C) MICROSOFT" replaced the snowy image of Napalmolive Madge soaking in detergent on the TV screen. No devil. No imps. Somewhere deep down inside, I noticed a little pang of disappointment. Too bad - now I'd have to learn to use the blasted thing.

Since that time, I have watched two other people overcome their fear of Color Computers. Both are women in their later years. One's my mother, the other is her sister. Mother started out by buying the machine (and a double disk drive, a tape machine, a passel of software, and a printer that cost more than all that put together!) Then she kept finding it impossible to find the time to use it - while the correspondence piled up, the newspaper clippings remained unfiled, and the games stayed in their cassettes, unplayed. It took her about six months to realize the real problem - and now she's anxious to get the machine back. Sez she has to have it - she "loaned" it to me, at one point. Gave up the whole system except the printer. I put it to work the next day at the hands of an assistant. John now has reserved his own CoCo system, but doesn't want to give hers up until his arrives. I guess I have to agree with him, in a way. He's revolutionized the way the company sells machines with a few simple text files he created in a couple of days.

The other is an aunt. She used to entertain the same sort of imaginings that I once used to justify my ignorance. Then one of her daughters wanted to learn music in the worst way, but with no money for college. The visit my cousin had with me (she shares none of her mother's fears) convinced her that a copy of Musica 2 by Speech Systems, plus a CoCo to run it on, would be exactly the right tool for the job. Next thing I heard, Auntie is out to a computer club meeting, buying copies of Russ Walter's Secret Guide To Computers for her own education. Life is funny.

Until next time...

Pleasant PL/9

Lane P. Lester
413 Woodland Circle
Lynchburg, VA 24502

The topic of this installment about the PL/9 compiler/language is the set of PL/9 keywords. Graham Trott's basic philosophy behind the use of keywords is that you ought to have all the words you need to do anything, but no more than are absolutely necessary. The most important reason for taking this approach is the need to keep the co-resident editor and compiler as small as possible. This maximizes the size of the program that can be developed in memory. Another benefit of this philosophy is that it doesn't take much study to get a handle on the compact vocabulary. To master all of the ways it may be effectively used, however, is a continuing adventure I expect to enjoy for a long time. The following is a list of the keywords with a brief explanation of each. The words are given in upper case, although case is completely optional.

ORIGIN $nnnn - Location of object code; same as assembly's ORG.

STACK $nnnn - Location of stack; essentially the same as assembly's LDS.

DPAGE = $nn - sets direct page register. I haven't used this one yet since my heritage is FORTRAN and BASIC rather than assembly.

MATHS $nnnn - Loads the arithmetic routines at location $nnnn.

AT $nnnn: tt aa - Assigns the name aa as type tt (BYTE, INTEGER or REAL) at location $nnnn. Designed primarily for hardware and STAR-DOS (or FLEX) addresses, but mighty nice for looking at variables when your program bombs.

GLOBAL tt aa - Assigns the name aa as a global variable (available to all procedures) of type tt.

CONSTANT aa = nn - Assigns the name aa as a constant nn which may be a byte or an integer.

BYTE aa nn... - These next three are like constants, and are often used to include data tables since a list of numbers can follow each name aa and be read as a subscripted variable.

INTEGER aa nn... -

REAL aa nn...

INCLUDE aa - When the compiler encounters this it reads in disk file aa, which is in PL/9 source, and compiles it as part of the program.

The above are generally put at the beginning of the program before any procedures, which are the individual modules or subroutines of a PL/9 program. The last procedure is where execution starts, and any procedure can call any previously-described procedure. The following are used within procedures.

PROCEDURE aa1 (tt2 aa2): tt3 aa3 - First line of a procedure. It is called by the name aa1 being included in a later procedure. It may have passed to it values named aa2 of type tt2. It will have local variables aa3 of type tt3.

ENDPROC aa - Signals the end of a procedure and may return a value aa.

RETURN aa - Although essentially the same as ENDPROC, this is a statement that will turn the stomach of a purist in structured programming. Whereever it is encountered in a procedure it ends the procedure and returns control to the calling procedure. It may return the value of aa. RETURN is usually invoked when some condition is met.

BEGIN, END - These are used to enclose a group of statements that are to be treated as a single unit.

IF expression

THEN statement

ELSE statement - Just like BASIC.

IF expression

CASE nn1 THEN statement1

CASE nn2 THEN statement2...

ELSE nnx THEN statementx - Provides for more than two possibilities.

WHILE condition statement - While the condition is true the statement will be repeatedly executed. A BEGIN-END can be used to control a group of statements.

REPEAT statement UNTIL condition - This differs from WHILE in that the statement will always be executed at least once.

REPEAT statement FOREVER - Just like it says.

AND, OR, EOR - These provide bitwise setting and clearing. ".AND", ".OR", ".EOR" - Logical operators useful in multiple tests in IF-THEN, IF-CASE, WHILE, and REPEAT-UNTIL statements. The quotation marks are not used; they are here for the benefit of the TSC Text Processor, which uses periods for formatting commands.

BREAK - Escapes from the current WHILE or REPEAT loop.

GOTO aa - A particularly nauseating example of the freedom that PL/9 provides the programmer. It transfers control back to some earlier point in the procedure at label aa.

CALL aa (or) $nnnn - Allows the calling of a subroutine outside of the PL/9 program. This allows the simple use of STAR-DOS (or FLEX) system routines, for example.

JUMP aa (or) $nnnn - Transfers control of the program.

GEN $nn... - Hex codes that follow GEN are inserted directly into the program and provide a way to use assembly routines... if for some reason you want to.

ASMPROC aa - Defines a procedure composed of just GEN statements.

ACCA = $nn
ACCB = $nn
CCR = $nn
ACCD = $nnnn

XREG = $nnnn - These four allow you to store values directly in the respective registers of the 6809. They also allow tests for particular contents.

RESET, NMI, FIRQ, IRQ, SWI, SWI2, SWI3 - allow the integration of interrupt service routines into PL/9 programs.

Because I do the grocery shopping for our family, this provided me with the inspiration for this article's PL/9 program. While I can handle the grocery store chore, Gail is in charge of the kitchen, so she needs to be able to provide me with the requisite shopping list. The hardest part of setting up this application was writing down a list of the items we buy to serve as a master file. What makes this whole enterprise worth while was the fact that the items are listed in the order I encounter them during my trek through the store.

GROC.CMD provides two functions, the first of which is to print in three columns on one page the 194 items of the master list. Gail circles the numbers on this list of the items she wants me to buy in my next shopping trip. She also writes in the quantities for anything more than one, plus any comments that may be appropriate to be sure I get what is needed. Then it's time to use the second function of GROC, preparing the shopping list. This involves entering the numbers of the items, and for the quantities, the SHIFT key plus the quantity. Yes, the permitted quantities are just 2 to 9, but what do you want for a free program? Seriously, I find this completely adequate, and it made the programming a bit simpler. To add in Gail's comments, I precede them with a ",". Entering an "S" starts the printout of the shopping list, and a typical entry would look like this:
4 TOMATO, LARGE ONES

```
/* GROC, Creates a shopping list from a database of items usually
 * purchased, sorted in the order encountered in thelstore.
 * Item and comment fields are limited to 32 characters.
 */
STACK = $07FF;

GLOBAL INTEGER ITEMS, TOTAL;

BYTE BLANKS "      ";

CONSTANT BLANK = $20, FORM_FEED = $0C, BELL = $07, ON = -1, OFF = 0;

AT $1200: BYTE LIST(6600);      /* The numbers in parentheses are */
AT $2C00: BYTE ITEM(3300);      /* not necessary, but can be      */
AT $3900: BYTE COMMENT(3300);   /* helpful for debugging and      */
AT $4600: BYTE QUANTITY(3300);  /* maintenance.                   */
AT $CB40: BYTE FCB(320);
AT $CC09: BYTE PAUSE;
AT $CC14: INTEGER LINE_BUFFER_POINTER;

INCLUDE IOSUBS;  /* I/O Procedures */
INCLUDE DOSSUBS; /* STAR-DOS (or FLEX) Procedures */
INCLUDE STRSUBS; /* String-manipulating Procedures */

PROCEDURE PRINT_CHAR (BYTE CHAR);
ACCA = CHAR;
CALL $CCE4;    /* $CCE4 = PRINTER, $CD18 = SCREEN */
ENDPROC;

PROCEDURE PRINT_STRING (BYTE .STRING);
WHILE STRING
   BEGIN
      PRINT_CHAR(STRING);
      .STRING = .STRING + 1; /* "." = pointer */
   END;
ENDPROC;


PROCEDURE PRINT_ITEM (INTEGER N: BYTE .ITEM: BYTE I, NUMBER(4);
NUMBER(0) = N / 100 + $30;
IF NUMBER(0) = $30 THEN NUMBER(0) = BLANK;
N = N - N / 100 * 100;
NUMBER(1) = N / 10 + $30;
IF NUMBER(0) = $20 .AND NUMBER(1) = $30 THEN NUMBER(1) = BLANK;
NUMBER(2) = N - N / 10 * 10 + $30;
NUMBER(3) = 0;
PRINT_STRING(.NUMBER); PRINT_CHAR('.'); PRINT_CHAR(BLANK);
I = 0;
REPEAT
   IF ITEM = 0
      THEN PRINT_CHAR(BLANK);
      ELSE PRINT_CHAR(ITEM);
   .ITEM = .ITEM + 1;
   I = I + 1;
UNTIL I = 33;
PRINT_STRING(.BLANKS);
ENDPROC;


PROCEDURE CLEAR_VECTORS: INTEGER I;
I = 0;
REPEAT
   ITEM(I) = 0;
   COMMENT(I) = 0;
   QUANTITY(I) = 0;
   I = I + 1;
UNTIL I = 3300;
ENDPROC;

PROCEDURE CHECK_ERROR(BYTE .FCB);
IF FCB(1) THEN
   BEGIN
      REPORT_ERROR(.FCB);
      DOS; /* Warm start of STAR-DOS or FLEX */
   END;
ENDPROC;

PROCEDURE READ_GROCERY_FILE: INTEGER I;
LINE_BUFFER_POINTER = "GROCRIES.DAT"; GET_FILENAME(.FCB);
OPEN_FOR_READ(.FCB); CHECK_ERROR(.FCB);
I = 0;
WHILE FCB(1) <> 8
   BEGIN
      LIST(I) = READ(.FCB);
      IF LIST(I) = CR
         THEN
            REPEAT
               LIST(I) = 0;
               I = I + 1
            UNTIL I \ 33 = 0;
         ELSE
            I = I + 1;
   END;
CLOSE_FILE(.FCB);
TOTAL = I / 33;
ENDPROC;

PROCEDURE PRINT_MASTER_LIST: INTEGER I, X, Y;
PRINT_CHAR($0F); /* EPSON COMPRESSED FONT */
X = TOTAL / 3;
Y = X + X;
I = 0;
REPEAT
   PRINT_ITEM(I,.LIST(I*33));
   PRINT_ITEM(I+X,.LIST((I+X)*33));
   PRINT_ITEM(I+Y,.LIST((I+Y)*33));
   PRINT_CHAR(CR); PRINT_CHAR(LF);
   I = I + 1;
UNTIL I = X;
PRINT_CHAR(FORM_FEED);
PRINT_CHAR($12); /* EPSON NORMAL FONT */
ENDPROC;



PROCEDURE ENTER_NUMBERS:
   BYTE BUFFER(32): INTEGER NUMBER, I, J, .LIST_ITEM;
PRINT("Enter numbers for items, shifted numbers for quantities (2-9).\N");
PRINT("Precede comments with "",", and enter ""S"" to stop.\N");
ITEMS = 0;
J = -33;
```

```
REPEAT
   INPUT(.BUFFER,32);
   CRLF;
   IF BUFFER = 'S .OR BUFFER = 's
      THEN RETURN;
      ELSE
         IF BUFFER = ',
            THEN STRCOPY(.COMMENT(I),.BUFFER);
            ELSE
               IF BUFFER < ',
                  THEN
                     BEGIN
                        BUFFER = BUFFER + $10;
                        STRCOPY(.QUANTITY(I),.BUFFER);
                     END;
                  ELSE
                     BEGIN
                        I = I + 33;
                        ITEMS = ITEMS + 1;
                        NUMBER = 0;
                        J = 0;
                        WHILE BUFFER(J) <> 0
                           BEGIN
                              NUMBER = NUMBER * 10;
                              NUMBER = NUMBER + BUFFER(J) - $30;
                              J = J + 1;
                           END;
                        .LIST_ITEM = .LIST + NUMBER * 33;
                        STRCOPY(.ITEM(I), .LIST_ITEM);
                     END;
FOREVER;
ENDPROC;

PROCEDURE PRINT_SHOPPING_LIST: INTEGER I, .POINTER;
I = 0;
WHILE ITEMS
   BEGIN
      IF QUANTITY(I) = 0 THEN QUANTITY(I) = BLANK;
      PRINT_STRING(.QUANTITY(I));
      PRINT_CHAR(BLANK);
      PRINT_STRING(.ITEM(I));
      PRINT_STRING(.COMMENT(I));
      PRINT_CHAR(CR); PRINT_CHAR(LF);
      I = I + 33;
      ITEMS = ITEMS - 1;
   END;
PRINT_CHAR(FORM_FEED);
ENDPROC;

PROCEDURE MAIN: BYTE CHAR;
CALL $CCC0; /* INITIALIZE PRINTER */
PAUSE = OFF;
PRINT("\N\NGrocery List");
READ_GROCERY_FILE;
CLEAR_VECTORS;
REPEAT
   PUTCHAR(BELL);
   PRINT("\N\NPress number of desired function\N\N");
   PRINT("1. Prepare shopping list.\N2. Print master list.\N");
   PRINT("3. Return to DOS.\N");
   REPEAT
      CHAR = GETCHAR;
   UNTIL CHAR > '0 .AND CHAR < '4;
   CRLF;
   IF CHAR
      CASE '1 THEN
         BEGIN
            ENTER_NUMBERS;
            PRINT_SHOPPING_LIST;
         END;
      CASE '2 THEN PRINT_MASTER_LIST;
UNTIL CHAR = '3;
PAUSE = ON;
```

Guarantee 680x Operation within Specs

Guarantee 680X Microprocessor

Controlled Computers Operation With

Respect to Processor Specifications

MR. HANS D. SCHMITZ, JR.
ELECTRICAL DESIGN ENGINEER
1286 HI-VIEW DRIVE
SOUTHAMPTON, PA 18966

All computers have operating parameters, the best case and worst case conditions determine these parameters and provide guaranteed proper operation reliability within their parameters. Computer designs which implement the 680X microprocessors as with all designs must be designed for guaranteed proper operation with respect to microprocessors specifications. The processors design specifications which must be obeyed are the worst case conditions over which it is specified to be guaranteed to reliably function.

Upon recently designing a 6809E up computer system, I encountered that interfacing to non-68XX peripherals and designing large systems controlled by 680X ups imposed an inherent conflict to the processor when buffering the data bus. The requirement for data bus buffers imposes the conflict of violating the processors worst case read-data hold time which the processor requires, and hence the processor and thus the entire computer system is not guaranteed for operation over any parameters. I have reviewed many 680X up controlled computers and have found approximately 90% in violation of this very important, easily overlooked, but crucial specification. Although systems may function within typical operating parameters (i.e. ambient room temperature environments and typical electrical conditions), there is no guarantee of any degree of reliability.

The 680X family of microprocessors have capacitive data buses internally, which helps the read data hold time to varying degrees. but in order to guarantee proper operation of the processor and entire computer system, all best-case, worst-case specifications should be strictly obeyed through proper logic and interface circuitry. Following are some solutions to the often disobeyed read data hold time specification for typical bus operations and interface requirements.

All of the following solutions will provide guaranteed proper operation of the data bus and will thus provide guaranteed reliable operation of the entire computer system. All of the following solutions provide proper logic circuitry for data bus operation, in addition some solutions provide interface specifications which enable interfacing to virtually all data bus requirements including industry bus standards. These solutions are able to correct all existing 680X processor computer systems which currently have this conflict, with only minor logic circuitry modifications.

A. Solution 1 is an inexpensive solution which supplies the correct data bus design for only 6809, 09E processor systems.

Solution 1's only disadvantage is the requirement for all the peripherals to have fast read data access times ranging within its restricting timing specifications.

B. Solution 2 is an inexpensive solution which supplies the correct data bus design for all 680X family processor systems.

Solution 2 has no apparent disadvantages, the required select logic and buffer/latch hardware logic circuitry are inexpensive industry standard I.C.'s, and this solution has many varieties of available bus operation and interface specifications complying to virtually all existing 680X processor systems, and thus fulfills virtually every possible data bus requirement and hence all existing peripherals.

Both solutions also provide varying bus operations and interface specifications which supply compliance to varying industry bus standards, accordingly the solution selected will be decided respectively.

Following are the technical specifics for both solutions 1, and 2.

SOLUTION 1:

Solution 1 provides an inexpensive correction for any of the 6809, 09E processor computers by using the direct connection wires for real time data transfer, and the data buffer as a latch when the buffer is enabled. Data buses desiring to implement this solution must operate within table 0's timing specifications. Table 5 shows the description of each of the referred times from table 0.

U1 is a bi-directional buffer which has proper output voltages which are guaranteed to properly operate within processor data bus input voltage requirements.

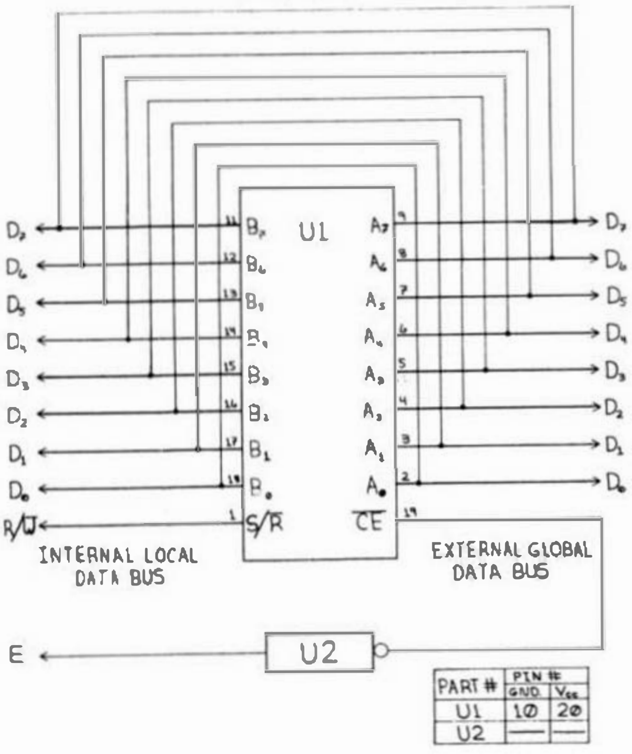
U2 is an inverting ttl compatible delay line which requires a minimum delay of t_{DLY1} holding the valid latched read data a minimum of the duration of the processors required t_{DHR}. U2 must avoid excessive delays so as to avoid data bus contention with the next following bus cycle in sequence. The digital delay line U2 requires time delays as noted in chart 0.

Figure 0 is solution 1's schematic diagram which will guarantee proper data hold time for all 6809, 09E processors only.

This solution (1) requires that on a processor read cycle; the logic circuitry's maximum delay times are such that the valid read data is on the local data bus before or consecutively with "E" clock rise above V_L to avoid contention when the data buffer emulates a latch.

This solution (1) on a write cycle holds valid data on the global data bus a minimum of the least of $t_{AH}+t_{U13}$ and $t_{DLY1}+t_{U11}$ after "E" fall edge below V_H, which supplies compliance to bus operations and interface specifications which require valid write

FIGURE Ø



data being held on the data bus greater than t_0 after the falling edge of the "E" clock.

The minimum time delay for $U2(t_{DLY1})$ is the time which guarantees the processors valid read data hold time after "E" clock fall edge $=V_L$.

The maximum time delay for $U2(t_{DLY2})$ is the time which avoids bus contention with the next following bus cycle in sequence.

This solution (1) requires that for the 6809, 09E the logic circuitry timing specifications guarantee strict read data setup times where $t_{DDQ}+t_{U14}$ is less than or equal to $t_{E1}-t_{AVS2}-t_{Qr2}+t_{DLY1}$ to the processor clocks. This solution (1) requires that for the 6809E the processor clock signals from clock generating circuitry guarantee strict read data setup times where $t_{DDQ}+t_{U14}$ is less than or equal to $t_{E1}-t_{BQ12}-t_{Qr2}+t_{DLY1}$.

SOLUTION 2:

This solution (2) will provide proper data communication logic for many variations of interface. Tables 1-4 show the variations for each 680X processor. The tables also show the data bus operation and interface advantages to determine which solution is appropriate for continued operatioo with existing peripherals. These solutions (2) provide guaranteed proper operation solutions to four different bus operation and interface specifications, which provide simple selection for the proper interface constraints which enable interfacing to existing peripherals. Table 5 shows the description of each of the referred times from tables 1-4.

Three configurations of buffer/latch select logic are supplied for varying options of bus operation and interface requirements.

U_{RD} and U_{WR} are either mono-directional buffers or latches depending on the bus operation and interface specifications required, both of which have bus operating specifications (voltage, current, and capacitance) which are guaranteed to properly operate within the processors data bus specifications.

U_{A1} and U_{B1} are non-inverting ttl compatible delay lines which buffer/hold valid read data a minimum of the duration of the processors required t_{DHR}. U_{A1} and U_{B1} must avoid excessive delays so as to avoid data bus contention with the next following bus cycle in sequence.

U_{A2}, U_{A3}, U_{B2}, U_{B3}, U_{B4}, and U_{C1} are the buffer/latch select logic gates which decode which of U_{RD} (read buffer/latch) and U_{WR} (write buffer/latch) to enable.

Tables 1-4 show the various bus operation and interface specifications which are available with solution 2.

Figure 1 is solution 2's schematic diagram which will guarantee proper data hold time for all 680X processors.

In conclusion, implementation of any of the above solutions will provide correct data bus design, and guaranteed reliable operation of all 680X processor computer systems.

FIGURE 1

BUFFER/LATCH SELECT LOGIC

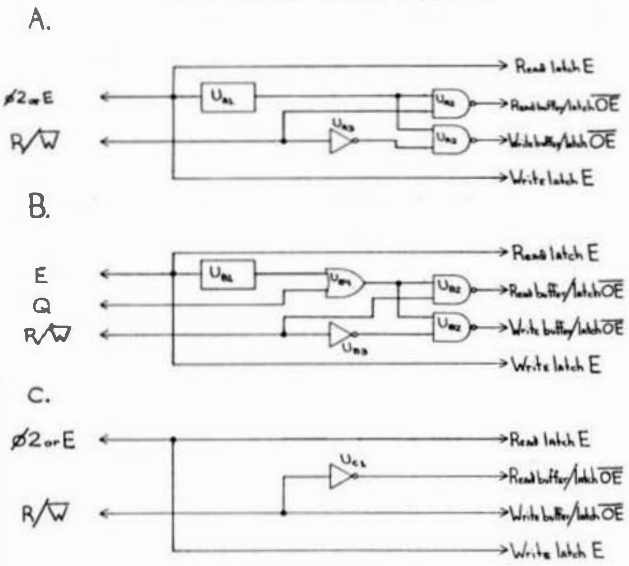
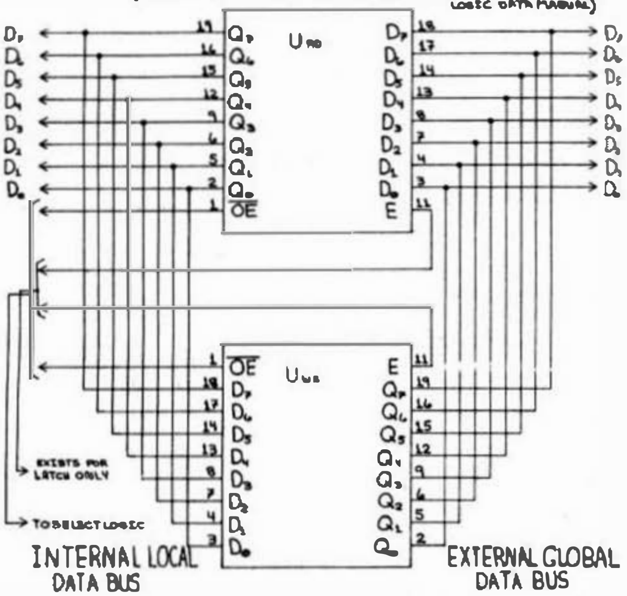


FIGURE 2

(PIN OUT IS FOR '373)

(PIN OUT FOR '244 CAN BE OBTAINED FROM SIGNETICS 1982 TTL LOGIC DATA MANUAL)



PART #	PIN # GND.	PIN # Vcc
U_{RD}	10	20
U_{WR}	10	20

CHART 0

processor type	t_{DLY1} (min.)	t_{DLY2} (max.)	unit
6809	$t_{Ef2}+t_{DHR}-t_{U11}$	$t_{AVS1}+t_{Qr1}-t_{U12}$	ns
6809E	$t_{Ef2}+t_{DHR}-t_{U11}$	$t_{EQ1}+t_{Qr1}-t_{U11}$	ns

TABLE 0

soln. #	up type	select logic	U_{RD}	U_{WR}	data bus, and interface specs. due to U_{RD}	due to U_{WR}
1	6809	N/A	'245		Read data real time: Read data is real-time during "E"=V_L. Read data latched: Read data is latched on "E" rise above V_L until a min. of t_{DHR} after "E" fall=V_L. Peripherals are req'd to supply valid read data on the global data bus, where the earliest time is consecutively with or before $t_{AVS}+t_{Qr1}-t_{DLY2}-t_{U12}$ leading "Q" rise=V_H, and also holding valid read data a min. of before or consecutively with $t_{U14}-t_{DLY1}$ leading the rise edge of "E" above V_L.	Write data real time: Write data is real-time during "E"=V_L. Write data latched: Write data is latched on "E" rise above V_L until a min. of t_{DLY1} after "E" fall less than V_H. Peripherals are req'd to store/latch up valid write data consecutively with or within $t_{DLY2}+t_{U15}$ after "E" rise=V_H, and also consecutively with or before the greater of $t_{DLY1}+T_{U11}$ and t_{DHW} after "E" fall edge=V_L.

TABLE 1.0

soln. #	up type	select logic	U_{RD}	U_{WR}	data bus, and interface specs. due to U_{RD}	due to U_{WR}
2	6800 (only where c2=DBE)	A	'244	'244	read data real time: up requires read data on the local bus a min. of t_{H1} after "c2" fall edge=V_L. Peripherals are req'd to hold valid read data on the global bus a min. of t_{H1} after "c2" fall edge=V_L. Select logic must insure a delay of $t_{c2}+t_{H1}$ from $t_{UA11}+t_{UA21}+t_{URD3}$.	write data real time: up holds valid write data on the local bus a min. of t_{H2} after "DBE" fall edge=V_L. Select logic must insure a min. delay of $t_{DBEf}+t_{H2}$ from $t_{UA11}+t_{UA21}+t_{UWR3}$. Peripherals are req'd to store/latch up write up write data within t_{H2} after "DBE" fall edge=V_L.

TABLE 1.1

soln. #	up type	select logic	U_{RD}	U_{WR}	data bus, and interface specs. due to U_{RD}	due to U_{WR}
2	6800 (only where c2=DBE)	A	'373	'373	Read data latched: up requires read data on the local bus a min. of t_{H1} after "c2" fall edge=V_L. Peripherals are req'd to hold valid read data on the global data bus a min. of t_O after the fall edge of "c2"=V_L; which is when the latch latches the read data. Select logic is req'd to provide a min. elay of $t_{c2f}+t_{H1}$ from $t_{UA11}+t_{UA21}+t_{URD3}$.	Write data latched: up holds valid write data on the local bus a min. of t_{H2} after "DBE" fall edge=V_L. Peripherals are req'd to store/latch up write data within $t_{UA11}+t_{UA21}+t_{UWR3}$ after "DBE" fall edge=V_L; which is the time to when the latch is disabled from driving valid data on the global data bus. Select logic is req'd to insure a min. delay of $t_{UA11}+t_{UA21}+t_{UWR3}$ which is dependant on the peripheral requirements.

The combinations of read, and write buffer/latch select logic '244,'373 and '373,'244 have the same respective parameters previously described.

TABLE 1.2

soln. #	up type	select logic	U_{RD}	U_{WR}	data bus, and interface specs. due to U_{RD}	due to U_{WR}
2	6800 (only where c2=DBE)	C	'244	'244	Read data r l time: up requires read data on the local bus a min. of t_{H1} after "c2" fall edge=V_L. Peripherals are req'd to hold valid read data on the global data bus a min. of t_{H1} after the fall edge of "c2"=V_L. Select logic is req'd to insure the read ata specs. to be obeyed such that $t_{AD}+t_{UC11}+t_{URD2}$ insures the processors read cycle requirements.	Write data real time: up holds valid write data on the local bus a min. of t_{H2} after "DBE" fall edge=V_L. Peripherals are req'd to store/latch up write data ithin t_{H2} after "DBE" fall edge=V_L. Select logic is req'd to insure the read data buffer to be disabled before "DBE"="c2" rise=V_H by insuring $t_{AD}+t_{UC11}$ to be less than t_{c21} to avoid data bus contention with a possible following write cycle next in sequence.

TABLE 1.3

soln.	up	select	U_{RD}	U_{WR}	data bus, and	interface specs.
#	type	logic			due to U_{RD}	due to U_{WR}
2	6800 (only where ø2=DBE)	C	'373	'373	Read data latched. up requires read data on the local bus a min. of t_{H1} after "ø2" fall edge→V_L. Peripherals are req'd to hold valid read data on the global data bus a min. of t_D after the fall edge of "ø2"→V_L; which is when the latch latches the read data.	Write data latched: up holds valid write data on the local bus a min. of t_{H2} after "DBE" fall edge→V_L. Peripherals are req'd to store/latch up write data within t_{AH}+t_{UWR3} after "ø1" fall edge→V_L.

The combinations of read, and write buffer/latch select logic '244,'373 and '373,'244 have the same respective parameters previously described.

TABLE 2.0

soln.	up	select	U_{RD}	U_{WR}	data bus, and	interface specs.
#	type	logic			due to U_{RD}	due to U_{WR}
2	6802 6808 6802NS	A	'244	'244	Read data real time: up requires read data on the local bus a min. of t_{H1} after "E" fall edge→V_L. Peripherals are req'd to hold valid read data on the global data bus a min. of t_{H1} after "E" fall edge→V_L. Select logic is req'd to insure a min. delay of t_{Ef2}+t_{H1} from t_{UA11}+t_{UA21}+t_{URD3}.	Write data real time: up holds valid write data on the local bus a min. of t_{H2} after "E" fall edge→V_L. Select logic is req'd to insure a min. delay of t_{Ef2}+t_{H2} from t_{UA11}+t_{UA21}+t_{UWR3}. Peripherals are req'd to store/latch up write data within t_{H2} after "E" fall edge→V_L.

TABLE 2.1

soln.	up	select	U_{RD}	U_{WR}	data bus, and	interface specs.
#	type	logic			due to U_{RD}	due to U_{WR}
2	6802 6808 6802NS	A	'373	'373	Read data latched: up requires read data on the local bus a min. of t_{H1} after "E" fall edge→V_L. Peripherals are req'd to hold valid read data on the global data bus a min. of t_D after the fall edge of "E"→V_L; which is when the latch latches the read data. Select logic is req'd to provide a min. delay of t_{Ef2}+t_{H1} from t_{UA11}+t_{UA21}+t_{URD3}.	Write data latched: up holds valid write data on the local bus a min. of t_{H2} after "E" fall edge→V_L. Peripherals are req'd to store/latch up write data within t_{UA11}+t_{UA21}+t_{UWR3} after "E" fall edge→V_L; which is the time to when the latch is disabled from driving valid data on the global data bus. Select logic is req'd to insure a min. delay of t_{UA11}+t_{UA21}+t_{UWR3} which is dependant on the peripheral requirements.

The combinations of read, and write buffer/latch select logic '244,'373 and '373,'244 have the same respective parameters previously described.

TABLE 2.2

soln.	up	select	U_{RD}	U_{WR}	data bus, and	interface specs.
#	type	logic			due to U_{RD}	due to U_{WR}
2	6800 6808 6802NS	C	'244	'244	Read data real time: up requires read data on the local bus a min. of t_{H1} after "E" fall edge→V_L. Peripherals are req'd to hold valid read data on the global data bus a min. of t_{H1} after the fall edge of "E"→V_L. Select logic is req'd to insure up the read data specs. to be obeyed such that t_{AH}+t_{UC11}+t_{URD2} insures the processors read cycle requirements.	Write data real time: up holds valid write data on the local bus a min. of t_{H2} after "E" fall edge→V_L. Peripherals are req'd to store/latch up write data within t_{H2} after "E" fall edge→V_L. Select logic is req'd to insure the read data buffer to be disabled before "E" rise→V_H by insuring t_{AH}+t_{UC11} to be less than t_{E1} to avoid data bus contention with a possible following write cycle next in sequence.

TABLE 2.3

soln.	up	select	U_{RD}	U_{WR}	data bus, and	interface specs.
#	type	logic			due to U_{RD}	due to U_{WR}
2	6800 6808 6802NS	C	'373	'373	Read data latched: up requires read data on the local bus a min. of t_{H1} after "E" fall edge→V_L. Peripherals are req'd to hold valid read data on the global data bus a min. of t_D after the fall edge of "E"→V_L; which is when the latch latches the read data.	Write data latched: up holds valid write data on the local bus a min. of t_{H2} after "E" fall edge→V_L. Peripherals are req'd to a ore/latch up write data within t_{AH}+t_{UWR3} after "E" fall edge→V_L.

The combinations of read, and write buffer/latch select logic '244,'373 and '373,'244 have the same respective parameters previously described.

TABLE 3.0

soln.	up	select	U_{RD}	U_{WR}	data bus, and	interface specs.
#	type	logic			due to U_{RD}	due to U_{WR}
2	6809	A	'244	'244	Read data real time: up requires read data on the local bus a min. of t_{DHR} after "E" fall edge→V_L. Peripherals are req'd to hold valid read data on the global data bus a min. of t_{DHR} after "E" fall edge→V_L. Select logic is req'd to insure a min. delay of t_{Ef2}+t_{DHR} from t_{UA11}+t_{UA21}+t_{URD3}.	Write data real time: up holds valid write data on the local bus a min. of t_{DHW} after "E" fall edge→V_L. Select logic is req'd to insure a min. delay of t_{Ef2}+t_{DHW} from t_{UA11}+t_{UA21}+t_{UWR3}, at which time valid write data no longer drives the global data bus. Peripherals are req'd to store/latch valid up write data within t_{DHW} after "E" fall edge→V_L.

TABLE 3.1

soln.	up	select	U_{RD}	U_{WR}	data bus, and	interfac specs.
#	type	logic			due to U_{RD}	due to U_{WR}
2	6809	A	'373	'373	Read data latched: up requires read data on the local bus a min. of t_{DHR} after "E" fall edge→V_L. Peripherals are req'd to hold valid read data on the global data bus a min. of t_D after the fall edge of "E"→V_L; which is when the latch latches the read data. Select logic is req'd to provide a min. delay of t_{Ef2}+t_{DHR} from t_{UA11}+t_{UA21}+t_{URD3}.	Write data latched: up holds valid write data on the local bus a min. of t_{DHW} after "E" fall edge→V_L. Peripherals are req'd to store/latch up write data within t_{UA11}+t_{UA21}+t_{UWR3} after "E" fall edge→V_L; which is the time to when the latch is disabled from driving valid data on the global data bus. Select logic is req'd to insure a min. delay of t_{UA11}+t_{UA21}+t_{UWR3} which is dependant on the peripheral re uirements.

The combinations of read, and write buffer/latch select logic '244,'373 and '373,'244 have the same respective parameters previously described.

TABLE 3.2

soln.	up	select	U_{RD}	U_{WR}	data bus, and	interface specs.
#	type	logic			due to U_{RD}	due to U_{WR}
2	6809	B	'244	'244	Read data real time: up requires read data on the local bus a min. of t_{DHR} after "E" fall edge→V_L. Peripherals are req'd to hold valid read data on the global data bus a min. of t_{DHR} after "E" fall edge→V_L. Select logic is req'd to insure the up's read data specifications are obeyed such that the read data buffer is disabled before "Q" rise→V_H by insuring t_{URD2}+t_{UB42}+t_{UB22} is less than t_{AVS1}-t_{AQ}.	Write data real time: up holds valid write data on the local bus a min. of t_{DHW} after "E" fall edge→V_L. Peripherals are req'd to latch or store valid write data within t_{UB11}+t_{UB41}+t_{UB21}+t_{DHW} after "E" fall edge→V_L, which the time at which the buffer is disabled from driving valid data on the global data bus. Select logic must insure the read data buffer is disabled from the current cycle before "Q" rise→V_H, of the next sequential cycle by insuring t_{Ef2}+t_{DHR} from t_{UB11}+t_{UB41}+t_{UB21}+t_{URD3} to avoid data bus contention with a possible write cycle next in sequence.

TABLE 3.1

soln. #	up type	select logic	V_{RD}	V_{WR}	data bus, and interface specs. due to V_{RD}	due to V_{WR}
2	6809	B	'373	'373	Read data latched: up requires read data on the local bus a min.of t_{DHR} after "E" fall edge=V_L. Peripherals are req'd to hold valid read data on the global data bus a min.of t_O after the fall edge of "E"=V_L; which is when the latch latches the read data. Sel ct logic is req'd to provide a min. delay of $t_{Ef2}+t_{DHR}$ from $t_{UB11}+t_{UB41}+t_{UB21}+t_{URD3}$.	Write data latched: up holds valid write data on the local bus a min. of t_{DHW} after "E" fall edge=V_L. Peripherals are req'd to store/latch up write data within $t_{UB11}+t_{UB41}+t_{UB21}+t_{UWR3}$ after "E" fall edge=V_L; which is the time to when the latch is disabled from driving valid write data on the global data bus. Select logic is req'd to insure a min. delay of $t_{UB11}+t_{UB41}+t_{UB21}+t_{UWR3}$ which is dependant on the peripheral requirements.

The combinatio s of read, and write buffer/latch select logic '244,'373 and '373,'244 have the same respective parameters previously described.

TABLE 3.4

soln. #	up type	select logic	V_{RD}	V_{WR}	data bus, and interface specs. due to V_{RD}	due to V_{WR}
2	6809	C	'244	'244	Read data real time: up requires read data on the local bus a min.of t_{DHR} after "E" fall edge=V_L. Peripherals are req'd to hold valid read data on the global data bus a min.of t_{DHR} after the fall edge of "E"=V_L. S lect logic is req'd to insure up's read data specs. to be obeyed such that $t_{AVS}-t_{AQ}+t_{UC11}+t_{URD3}$ insures the processors read cycle requirements.	Write data real time: up holds valid write data on the local bus a min. of t_{DHW} after "E" fall edge=V_L. Peripherals are req'd to store/latch up write data within t_{DHW} after "E" fall edge=V_L. Select logic is req'd to insure the read data buffer to be disabled before "E" rise=V_H by insuring $t_{URD}+t_{UC11}$ to be less than t_{AQ} to avoid data bus contention with a possible following write cycle next in sequence.

TABLE 3.5

soln. #	up type	select logic	V_{RD}	V_{WR}	data bus, and interface specs. due to V_{RD}	due to V_{WR}
2	6809	C	'373	'373	Read data latched: up requires read data on the local bus a min.of t_{DHR} after "E" fall edge=V_L. Peripherals are req'd to hold valid read data on the global data bus a min.of t_O after the fall edge of "E"=V_L; which is when the latch latches the read data.	Write data latched: up holds valid write data on the local bus a min. of t_{DHW} after "E" fall edge=V_L. Peripherals are req'd to store/latch up write data within $t_{AH}+t_{UWR3}$ after "E" fall edge=V_L.

The combinations of read, and write buffer/latch select logic '244,'373 and '373,'244 have the same respective parameters previously described.

TABLE 4.0

soln. #	up type	select logic	V_{RD}	V_{WR}	data bus, and interface specs. due to V_{RD}	due to V_{WR}
2	6809E	A	'244	'244	Read data real time: up requires read data on the local bus a min.of t_{DHR} after "E" fall edge=V_L. Peripherals are req'd to hold valid read data on the global data bus a min.of t_{DHR} after "E" fall edge=V_L. Select logic is req'd to insure a min.delay of $t_{Ef2}+t_{DHR}$ from $t_{UA11}+t_{UA21}+t_{URD3}$.	Write data real time: up holds valid write data on the local bus a min. of t_{DHW} after "E" fall edge=V_L. Select logic is req'd to insure a min.delay of $t_{Ef2}+t_{DHW}$ from $t_{UA11}+t_{UA21}+t_{UWR3}$, at which time valid write data no longer drives the global data bus. Peripherals are req'd to store/latch valid up write data within t_{DHW} after "E" fall edge=V_L.

TABLE 4.1

soln. #	up type	select logic	V_{RD}	V_{WR}	data bus, and interface specs. due to V_{RD}	due to V_{WR}
2	6809E	A	'373	'373	Read data latched: up requires read data on the local bus a min.of t_{DHR} after "E" fall edge=V_L. Peripherals are req'd to hold valid read data on the global data bus a min.of t_O after the fall edge of "E"=V_L; which is when the latch latches the read data. Select logic is req'd to provide a min. delay of $t_{Ef2}+t_{DHR}$ from $t_{UA11}+t_{UA21}+t_{URD3}$.	Write data latched: up holds valid write data on the local bus a min. of t_{DHW} after "E" fall edge=V_L. Peripherals are req'd to store/latch up write data within $t_{UA11}+t_{UA21}+t_{UWR3}$ after "E" fall edge=V_L; which is the time to when the latch is disabled from driving valid data on the global data bus. Select logic is req'd to insure a min. delay of $t_{UA11}+t_{UA21}+t_{UWR3}$ which is dependant on the peripheral requirements.

The combinations of read, and write buffer/latch select logic '244,'373 and '373,'244 have the same respective parameters previously described.

TABLE 4.2

soln. #	up type	select logic	V_{RD}	V_{WR}	data bus, and interface specs. due to V_{RD}	due to V_{WR}
2	6809E	B	'244	'244	Read data real time: up r quires read data on the local bus a min.of t_{DHR} after "E" fall edge=V_L. Peripherals are req'd to hold valid read data on the global data bus a min.of t_{DHR} after "E" fall edge=V_L. Select logic is req'd to insure the up's read data specifications are obeyed such that the read data buffer is disabled before "Q" rise=V_H by insuring $t_{URD2}+t_{UB42}+t_{UB22}$ to less than $t_{AVS1}-t_{AQ}$.	Write data real time: up holds valid write data on the local bus a min. of t_{DHW} after "E" fall edge=V_L. Peripherals are req'd to latch or sto e valid write data within $t_{UB11}+t_{UB41}+t_{UB21}+t_{DHW}$ after "E" fall edge=V_L, which the time at which the buffer is di abled from driving valid data on the global data bus. Select logic must insure the read data buffer is disabled from the current cycle before "Q" rise=V_H, of the next sequential cycle by insuring $t_{Ef2}+t_{DHR}$ from $t_{UB11}+t_{UB41}+t_{UB21}+t_{URD3}$ to avoid data bus contention with a possible write cycle next in sequence.

TABLE 4.3

soln. #	up type	select logic	V_{RD}	V_{WR}	data bus, and interface specs. due to V_{RD}	due to V_{WR}
2	6809E	B	'373	'373	Read data latched: up requires read data on the local bus a min.of t_{DHR} after "E" fall edge=V_L. Peripherals are req'd to hold valid read data on the global data bus a min.of t_O after the fall edge of "E"=V_L; which is when the latch latches the read data. Select logic is req'd to provide a min. delay of $t_{Ef2}+t_{DHR}$ from $t_{UB11}+t_{UB41}+t_{UB21}+t_{URD3}$.	Write data latched: up holds valid write data on the local bus a min. of t_{DHW} after "E" fall edge=V_L. Peripherals are req'd to store/latch up write data within $t_{UB11}+t_{UB41}+t_{UB21}+t_{UWR3}$ after "E" fall edge=V_L; which is the time to when the latch is disabled from driving valid write data on the global data bus. Select logic is req'd to insure a min. delay of $t_{UB11}+t_{UB41}+t_{UB21}+t_{UWR3}$ which is dependant on the peripheral requirements.

The combinations of read, and w ite buffer/latch select logic '244,'373 and '373,'244 have the same respective parameters previously described.

Continued on page 35

FINALLY !!

Now you can run

TSC XBASIC Programs Compiled to Asmb. Lang., under **OS-9™, CoCo OS-9, or FLEX™ with**

✫ K-BASIC

TELEX 558 414 PVT BTH

☎ (615)842-4600

South East Media

5900 Cassandra Smith Rd.
Hixson, TN 37343
for Information
call (615) 842-4801

CoCo OS-9™ FLEX™

SOFTWARE

K-BASIC under OS-9 and FLEX will now compile TSC BASIC, XBASIC, and XPC Source Code Files

K-BASIC now makes the multitude of TSC XBASIC Software available for use under OS-9. Transfer your favorite BASIC Programs to OS-9, compile them, Assemble them, and BINGO -- usable, multi-precision, familiar Software is running under your favorite Operating System!

K-BASIC (OS-9 or FLEX), including the Assembler

$199.00

==

SCULPTOR

Microprocessor Developments Ltd.'s Commercial Application Generator Program provides a FAST Commercial Application Development tool unavailable to the OS-9 and UniFLEX User before. Develop any Commercial Application in 20% of the normal required time; gain easy updating or customizing. **PLUS**, the Application can also be run on MS-DOS and Unix machines! **Sculptor** handles input validation, complex calculations, and exception conditions as well as the normal collecting, displaying, reporting, and updating information in an orderly fashion. Key fields to 160 bytes; unlimited record size; file size should be held to 17 million records. Utilizes **ISAM** File Structure and B-tree Key files for rapid access. Input and Output communication with other programs and files plus a library of ISAM routines for use with C Programs. Run-time included w/ the Development package; a compiled Application only needs a Run-time License. Additional charge for Networked Units. Prices for **Development Package/Run-time**. Discounts available for purchases of 5 or more Run-time Packages.

UniFLEX — $1450.00/337.00
OS-9 Lvl II -- $995.00/200.00

Unix -- up to $2535/675 -- Call for info!
IBM PC & compats -- $635.00/170.00

==

✫ **Basic09 Tour Guide**

by Dale Puckett -- An excellent Book on using OS-9. Oriented towards using the powerful **Basic09 Programming Language**, it also contains a lot of good information on **using** OS-9 in general.

Normally $18.95
Special ---- NOW only $16.00

✫ —— **FLEX Software** ----

TSC "Flex Utilities"	was $75.00,	**NOW** only **$65.00**
TSC "Sort Merge"	was $75.00,	**NOW** only **$65.00**
TSC "6809 Basic"	was $75.00,	**NOW** only **$65.00**
TSC "Extended Basic"	was $100.00,	**NOW** only **$90.00**
TSC "DeBug"	was $75.00,	**NOW** only **$65.00**
TSC "FLEX Diagnostics"	was $75.00,	**NOW** only **$65.00**
TSC "Text Processing System"	was $75.00,	**NOW** only **$65.00**
TSC "68000 Cross Assembler"	was $250.00,	**NOW** only **$199.95**

VISA MasterCard

** SHIPPING **
Add 2% U.S.A.
(min. $2.50)
Add 5% Surface Foreign
10% Air Foreign

South East Media
5900 Cassandra Smith Rd.
Hixson, TN 37343
info (615) 842-4801
CoCo OS-9™ FLEX™
SOFTWARE

Availability Legends —
F = FLEX, CCF = Color Computer FLEX
O = OS-9, CCO = Color Computer OS-9
U = UniFLEX
CCD = Color Computer Disk
CCT = Color Computer Tape

™FLEX is a trademark of Technical Systems Consultants
™OS9 is a trademark of Microware

!!! Please Specify Your Operating System & Disk Size !!!

☎ (615)842-4600 TELEX 558 414 PVT BTH

South East Media

5900 Cassandra Smith Rd.
Hixson, TN 37343

for information
call (615) 842-4601

CoCo OS-9™ FLEX™

SOFTWARE

PROGRAMMING

ASSEMBLERS

ASTRUK09 from **Southeast Media** -- A "Structured Assembler for the 6809" which requires the TSC **Macro Assembler**. F, CCF - $99.95

Macro Assembler for TSC -- The FLEX **STANDARD** Assembler.
Special -- CCF $35.00; F $50.00

OSM Extended 6809 Macro Assembler from **Lloyd I/O**. -- Provides local labels, Motorola S-records, and Intel Hex records; XREF. Generate OS-9 Memory modules under FLEX. FLEX, CCF, OS-9 $99.00

Relocating Assembler w/Linking Loader from TSC. -- Use with many of the C and Pascal Compilers. F,CCF $150.00

MACE, by Graham Trott from **Windrush Micro Systems** -- Co-Resident Editor and Assembler; fast interactive A.L. Programming for small to medium-sized Programs. F,CCF - $75.00

XMACE -- MACE w/ Cross Assembler for 6800/1/2/3/8 F,CCF - $98.00

TRUE CROSS ASSEMBLERS from **Computer Systems Consultants** -- Supports 1802/5, Z-80, 6800/1/2/3/8/11/HC11, 6804, 6805/HC05/146805, 6809/00/01, 6502 family, 8080/5, 8020/1/2/35/C35/39/40/48/C48/49/C49/50/8748/49, 8031/51/8751, and 68000 Systems. Assembler and Listing formats same as target CPU's format. Produces machine independent Motorola S-Text.
FLEX, CCF, OS-9, UniFLEX each - $50.00
any 3 - $100.00
the complete set w/ C Source (<u>except</u> the 68000 Source) - $200.00

XASM Cross Assemblers for FLEX from **Compusense Ltd.** -- This set of 6800/1/2/3/5/8, 6301, 6502, 8080/5, and Z80 Cross Assemblers uses the familiar TSC Macro Assembler Command Line and Soruce Code format, Assembler options, etc., in providing code for the target CPU's. Complete set, FLEX only - $150.00

CRASMB from Lloyd I/O -- 8-Bit Macro Cross Assembler with same features as OSM; cross-assemble to 6800/1/2/3/4/5/8/9/11, 6502, 1802, 8048 Sers, 80/85, Z-8, Z-80, TMS-7000 sers. Supports the target chip's standard mnemonics and addressing modes.
FLEX, CCF, OS-9 Full package -- $399.00

CRASMB 16.32 from Lloyd I/O -- Cross Assembler for the 68000.
FLEX, CCF, OS-9 $249.00

DISASSEMBLERS

SUPER SLEUTH from **Computer Systems Consultants** -- Interactive Disassembler; estremely POWERFUL! Disk File Binary/ASCII Examine/Change, Absolute or FULL Disassembly. XREF Generator, Label "Name Changer", and Files of "Standard Label Names" for different Operating Systems

Color Computer		SS-50 Bus (All w/ A.L. Source)
CCD (32K Req'd) Obj. Only	$49.00	F, $99.00
CCF, Obj. Only	$50.00	U, $100.00
CCF, w/Source	$99.00	O, $101.00
CCO, Obj. Only	$50.00.	

DYNAMITE + from **Computer Systems Center** -- Excellent standard "Batch Mode" Disassembler. Includes XREF Generator and "Standard Label" Files. Special OS-9 options w/ OS-9 Version.

CCF, Obj. Only	$100.00	CCO, Obj. Only	$ 99.95
F, " "	$100.00	O, " "	$150.00
		U, " "	$300.00

PROGRAMMING LANGUAGES

PL/9 from **Windrush Micro Systems** -- By Graham Trott. A combination Editor/Compiler/Debugger. Direct source-to-object compilation delivering fast, compact, re-entrant, ROM-able, PIC. 8 & 16-bit Integers & 6-digit Real numbers for all real-world problems. Direct control over ALL System resources, including interrupts. Comprehensive library support; simple Machine Code interface; step-by-step tracer for instant debugging. **500+** page Manual with tutorial guide. F, CCF - $198.00

WHIMSICAL from **Whimsical Developments** -- Now supports **Real Numbers**. "Structured Programming" WITHOUT losing the Speed and Control of Assembly Language! Single-pass Compiler features unified, user-defined I/O; produces ROMable Code; Procedures and Modules (including pre-compiled Modules); many "Types" up to 32 bit Integers, **6-digit Real Numbers**, unlimited sized Arrays (vectors only); Interrupt handling; long Variable Names; Variable Initialization; Include directive; Conditional compiling; direct Code insertion; control of the Stack Pointer; etc. Run-Time subroutines inserted as called during compilation. Normally produces 10% less code than PL/9. F and CCF - $195.00

C Compiler from **Windrush Micro Systems** by James McCosh. Full C for FLEX except bit-fields, including an Assembler. Requires the TSC Relocating Assembler if user desires to implement his own Libraries. F and CCF - $295.00

C Compiler from **Introl** -- Full C except Doubles and Bit Fields, streamlined for the 6809. Reliable Compiler; FAST, efficient Code. More UNIX Compatible than most.
F, CCF, and O - $375.00 U - $425.00

PASCAL Compiler from **Lucidata** -- ISO Based P-Code Compiler. Designed especially for Microcomputer Systems. Allows linkage to Assembler Code for maximum flexibility.
F and CCF 5" - $190.00 F 8" - $205.00

PASCAL Compiler from **OmegaSoft** (now **Certified Software**) — For the **PROFESSIONAL**; ISO Based, Native Code Compiler. Primarily for Real-Time and Process Control applications. Powerful; Flexible. Requires a "Motorola Compatible" Relo. Asmb. and Linking Loader.
F and CCF - $425.00 One Year Maint. - $100.00

K-BASIC from **LLOYD I/O** -- A "Native Code" **BASIC Compiler** which is now Fully TSC XBASIC compatible. The compiler compiles to Assembly Language Source Code. A NEW, streamlined, Assembler is now included allowing the assembly of LARGE Compiled K-BASIC Programs. Conditional assembly reduces Run-time package.
FLEX, CCF, OS-9 Compiler with Assembler - $199.00

CRUNCH COBOL from **Compusense Ltd.** -- Supports large subset of ANSII Level 1 COBOL with many of the useful Level 2 features. Full FLEX File Structures, including Random Files and the ability to process Keyed Files. Segment and link large programs at runtime, or implemented as a set of overlays. The System requires 56K and CAN be run with a single Disk System.
FLEX, CCF; Normally $199.00
Special Introductory Price (while in effect) -- $99.95

FORTH from **Stearns Electronics** -- A CoCo **FORTH** Programming Language. **Tailored to the CoCo!** Supplied on Tape, transferable to disk. Written in FAST ML. Many CoCo functions (Graphics, Sound, etc.). Includes an Editor, Trace, etc. Provides CPU Carry Flag accessibility, Fast Task Multiplexing, Clean Interrupt Handling, etc. for the "Pro". Excellent "Learning" tool!
Color Computer **ONLY** - $58.95

VISA MasterCard

** SHIPPING **
Add 2% U.S.A.
(min. $2.50)
Add 5% Surface Foreign
10% Air Foreign

South East Media
5900 Cassandra Smith Rd.
Hixson, TN 37343
info (615) 842-4601
CoCo OS-9™ FLEX™
SOFTWARE

Availability Legends —

F = FLEX,	CCF = Color Computer FLEX
O = OS-9,	CCO = Color Computer OS-9
U = UniFLEX	
	CCD = Color Computer Disk
	CCT = Color Computer Tape

™FLEX is a trademark of Technical Systems Consultants
™OS9 is a trademark of Microware

!!! Please Specify Your Operating System & Disk Size !!!

GENERAL

SOFTWARE DEVELOPMENT

BasicO9 XRef from Southeast Media -- This **BasicO9 Cross Reference Utility** is a BasicO9 Program which will produce a "pretty printed" listing with each line numbered, followed by a complete cross referenced listing of all variables, external procedures, and line numbers called. Also includes a **Program List Utility** which outputs a fast "pretty printed" listing with line numbers. Requires BasicO9 or RunB.
O & CCO obj. only -- $39.95; w/ Source - $79.95

Lucidata PASCAL UTILITIES (Requires LUCIDATA Pascal ver 3)

XREF -- produce a Cross Reference Listing of any text; oriented to Pascal Source.

INCLUDE -- Include other Files in a Source Text, including Binary; unlimited nesting capabilities.

PROFILER -- provides an Indented, Numbered, "Structogram" of a Pascal Source Text File; view the overall structure of large programs, program integrity, etc. Supplied in Pascal Source Code; requires compilation.

F, CCF --- **EACH Utility** 5" - $40.00, 8" - $50.00

DUB from Southeast Media -- A UniFLEX "basic" De-Compiler. Re-Create a Source listing from UniFLEX **Compiled** basic Programs. Works w/ ALL Versions of 6809 UniFLEX basic. U - $219.95

FULL **SCREEN FORMS DISPLAY** from **Computer Systems Consultants** -- TSC Extended BASIC program supports any Serial Terminal with Cursor Control or Memory-Mapped Video Displays; substantially extends the capabilities of the Program Designer by providing a table-driven method of describing and using Full Screen Displays.
F and CCF, U - $25.00, w/ Source - $50.00

DISK UTILITIES

OS-9 VDisk from Southeast Media -- For **Level I** only. Use the **Extended Memory** capability of your SWTPC or Gimix CPU card (or similar format DAT) for FAST Program Compiles, CMD execution, high speed inter-process communications (without pipe buffers), etc. - SAVE that System Memory. Virtual Disk size is variable in 4K increments up to 960K. Some Assembly Required.
-- **Level I ONLY** -- OS-9 obj. only - $79.95; w/ Source - $149.95

O-F from Southeast Media -- Written in **BASICO9** (with Source). includes: **REFORMAT**, a BASICO9 Program that reformats a chosen amount of an OS-9 disk to FLEX Format so it can be used normally by FLEX; and FLEX, a BASICO9 Program that does the actual read or write function to the special O-F Transfer Disk; user-friendly menu driven. Read the FLEX Directory, Delete FLEX Files, Copy both directions, etc. FLEX users use the special disk just like any other FLEX disk. O - $79.95

COPYMULT from Southeast Media -- Copy LARGE Disks to several smaller disks. FLEX utilities allow the backup of **ANY size disk** to any SMALLER size diskettes (Hard Disk to floppies, 8" to 5", etc.) by simply inserting diskettes as requested by **COPYMULT**. No fooling with directory deletions, etc. **COPYMULT.CMD** understands normal "copy" syntax and keeps up with files copied by maintaining directories for both host and receiving disk system. Also includes **BACKUP.CMD** to download any size "random" type file; **RESTORE.CMD** to restructure copied "random" files for copying, or recopying back to the host system; and **FREELINK.CMD** as a "bonus" utility that "relinks" the free chain of floppy or hard disk, eliminating fragmentation.

Completely documented Assembly Language Source files included.
ALL 4 Programs (FLEX, 8" or 5") $99.50

COPYCAT from Lucidata -- Pascal **NOT** required. Allows reading TSC Mini-FLEX, SSB DOS68, and Digital Research CP/M Disks while operating under FLEX 1.0, FLEX 2.0, or FLEX 9.0 with 6800 or 6809 Systems. COPYCAT will not perform miracles, but, between the program and the manual, you stand a good chance of accomplishing a transfer. Also includes some Utilities to help out. Programs supplied in **Modular Source Code** (Assembly Language) to help solve unusual problems.
F and CCF 5" - $50.00 F 8" - $65.00

☎ TELEX 558 414 PVT BTH

(615)842-4600

SOUTH EAST MEDIA

5900 Cassandra Smith Rd.
Hixson, TN 37343
for information
call (615) 842-4601

CoCo OS-9™ FLEX™
SOFTWARE

FLEX DISK UTILITIES from Computer Systems Consultants -- Eight (8) different Assembly Language (w/ Source Code) FLEX Utilities for every FLEX Users Toolbox: **Copy a File with CRC Errors; Test Disk** for errors; **Compare two Disks**; a fast **Disk Backup** Program; **Edit Disk Sectors; Linearize Free-Chain** on the Disk; print **Disk identification**; and **Sort** and **Replace** the Disk Directory (in sorted order). -- PLUS -- Ten XBASIC Programs including: A **BASIC Resequencer** with **EXTRAs** over "**RENUM**" like check for missing label definitions, processes Disk to Disk instead of in Memory, etc. Other programs **Compare, Merge, or Generate Updates** between two BASIC Programs, check BASIC **Sequence Numbers**, compare two **unsequenced** files, and 5 Programs for establishing a Master Directory of several Disks, and sorting, selecting, updating, and printing paginated listings of these files. A BASIC Cross-Reference Program, written in Assembly Language, which provides an X-Ref Listing of the Variables and Reserved Words in **TSC BASIC, XBASIC, and PRECOMPILER BASIC Programs**.
ALL Utilities **include Source** (either BASIC or A.L. Source Code).
F and CCF - $50.00
BASIC Utilities ONLY for UniFLEX -- $30.00

COMMUNICATIONS

CMODEM Telecommunications Program from Computer Systems Consultants, Inc. -- Menu-Driven; supports Dumb-Terminal Mode, Upload and Download in non-protocol mode, and the CP/M "Modem7" Christensen protocol mode to enable communication capabilities for almost any requirement. Written in "C".
FLEX, CCF, OS-9, UniFLEX; with complete Source - $100.00
without Source - $50.00

XDATA from Southeast Media -- A **COMMUNICATION** Package for the UniFLEX Operating System. Use with CP/M, Main Frames, other UniFLEX Systems, etc. Verifies Transmission using checksum or CRC; Re-Transmits bad blocks, etc. U - $299.99

GAME

RAPIER - 6809 **Chess** Program from **Southeast Media** -- Requires FLEX and Displays on **Any** Type Terminal. Features: Four levels of play. Swap side. Point scoring system. Two display boards. Change skill level. Solve Checkmate problems in 1-2-3-4 moves. Make move and swap sides. Play white or black. This is one of the strongest CHESS programs running on any microcomputer, estimated USCF Rating 1600+ (better than most 'club' players at higher levels). F and CCF - $79.95

VISA

-- SHIPPING --
Add 2% U.S.A.
(min. $2.50)
Add 5% Surface Foreign
10% Air Foreign

SOUTH EAST MEDIA
5900 Cassandra Smith Rd.
Hixson, TN 37343
info (615) 842-4601
CoCo OS-9™ FLEX™
SOFTWARE

Availability Legends ---
F = FLEX, CCF = Color Computer FLEX
O = OS-9, CCO = Color Computer OS-9
U = UniFLEX
CCD = Color Computer Disk
CCT = Color Computer Tape

™FLEX is a trademark of Technical Systems Consultants
™OS9 is a trademark of Microware

!!! Please Specify Your Operating System & Disk Size !!!

☎ (615)842-4600

TELEX 558 414 PVT BTH

SOUTH EAST MEDIA

5900 Cassandra Smith Rd.
Hixson, TN 37343
for information
call (615) 842-4601

CoCo OS-9™ FLEX™
SOFTWARE

WORD PROCESSING

SCREDITOR III from Windrush Micro Systems -- Powerful Screen-Oriented Editor/Word Processor. Almost 50 different commands; over 300 pages of Documentation with Tutorial. Features Multi-Column display and editing, "decimal align" columns (AND add them up automatically), multiple keystroke macros, even/odd page headers and footers, imbedded printer control codes, all justifications, "help" support, store common command series on disk, etc. Use supplied "set-ups", or remap the keyboard to your needs. Except for proportional printing, this package will DO IT ALL!

6800 or 6809 FLEX or SSB DOS, OS-9 - $175.00

STYLO-GRAPH from Great Plains Computer Co. -- A full-screen oriented WORD PROCESSOR -- (uses the 51 x 24 Display Screens on CoCo FLEX/STAR-DOS, or PBJ Wordpak). Full screen display and editing; supports the Daisy Wheel proportional printers.

NEW PRICES —> CCF and CCO - $99.95, F or O - $179.95, U - $299.95

STYLO-SPELL from Great Plains Computer Co. -- Fast Computer Dictionary. Complements Stylograph.

NEW PRICES —> CCF and CCO - $69.95, F or O - $99.95, U - $149.95

STYLO-MERGE from Great Plains Computer Co. — Merge Mailing List to "Form" Letters, Print multiple Files, etc., through Stylo.

NEW PRICES —> CCF and CCO - $59.95, F or O - $79.95, U - $129.95

STYLO-PAK — Graph + Spell + Merge Package Deal!!!

F or O - $329.95, U - $549.95

JUST from Southeast Media -- Text Formatter developed by Ron Anderson; for Dot Matrix Printers, provides many unique features. Output "Formatted" Text to the Display. Use the FPRINT.CMD supplied for producing multiple copies of the "Formatted" Text on the Printer INCLUDING IMBEDDED PRINTER COMMANDS (very useful at other times also, and worth the price of the program by itself). "User Configurable" for adapting to other Printers (comes set up for Epson MX-80 with Graftrax); up to ten (10) imbedded "Printer Control Commands". Compensates for a "Double Width" printed line. Includes the normal line width, margin, indent, paragraph, space, vertical skip lines, page length, page numbering, centering, fill, justification, etc. Use with ANY Editor. Supplied with "Structured Source" (Windrush PL/9); easy to see the flow of the program.

F and CCF - $49.95

SPELLB "Computer Dictionary" from Southeast Media -- OVER 120,000 words! Look up a word from within your Editor or Word Processor (with the SPH.CMD Utility which operates in the FLEX UCS). Or check and update the Text after entry; ADD WORDS to the Dictionary, "Flag" questionable words in the Text, "View a word in context" before changing or ignoring, etc. SPELLB first checks a "Common Word Dictionary", then the normal Dictionary, then a "Personal Word List", and finally, any "Special Word List" you may have specified. SPELLB also allows the use of Small Disk Storage systems.

F and CCF - $129.95

DATA BASE - ACCOUNTING

XDMS from Westchester Applied Business Systems -- Powerful DBMS; M.L. program will work on a single sided 5" disk, yet is F-A-S-T. Supports Relational, Sequential, Hierarchical, and Random Access File Structures; has Virtual Memory capabilities for Giant Data Bases. XDMS Level I provides an "entry level" System for defining a Data Base, entering and changing the Data, and producing Reports. XDMS Level II adds the POWERFUL "GENERATE" facility with an English Language Command Structure for manipulating the Data to create new File Structures, Sort, Select, Calculate, etc. XDMS Level III adds special "Utilities" which provide additional ease in setting up a Data Base, such as copying old data into new Data Structures, changing System Parameters, etc.

XDMS System Manual - $24.95   XDMS Lvl I - F & CCF - $129.95
XDMS Lvl II - F & CCF - $199.95
XDMS Lvl III - F & CCF - $289.95

ACCOUNTING PACKAGES -- Great Plains Computer Co. and Universal Data Research, Inc. both have Data Base and Business Packages written in TSC XBASIC for FLEX, CoCo FLEX, and UniFLEX.

MISCELLANEOUS

TABULA RASA SPREADSHEET from Computer Systems Consultants -- TABULA RASA is similar to DESKTOP/PLAN; provides use of tabular computation schemes used for analysis of business, sales, and economic conditions. Menu-driven; extensive report-generation capabilities. Requires TSC's Extended BASIC.

F and CCF, U - $50.00, w/ Source - $100.00

DYNACALC from Computer Systems Center -- Electronic Spread Sheet for the 6809.

F and SPECIAL CCF - $200.00, U - $395.00

FULL SCREEN INVENTORY/MRP from Computer Systems Consultants -- Use the Full Screen Inventory System/Materials Requirement Planning for maintaining inventories. Keeps item field file in alphabetical order for easier inquiry. Locate and/or print records matching partial or complete item, description, vendor, or attributes; find backorder or below stock levels. Print-outs in item or vendor order. MRP capability for the maintenance and analysis of Hierarchical assemblies of items in the inventory file. Requires TSC's Extended BASIC.

F and CCF, U - $50.00, w/ Source - $100.00

FULL SCREEN MAILING LIST from Computer Systems Consultants -- The Full Screen Mailing List System provides a means of maintaining simple mailing lists. Locate all records matching on partial or complete name, city, state, zip, or attributes for Listings or Labels, etc. Requires TSC's Extended BASIC.

F and CCF, U - $50.00, w/ Source - $100.00

DIET-TRAC Forecaster from Southeast Media -- An XBASIC program that plans a diet in terms of either calories and percentage of carbohydrates, proteins and fats (C P G%) or grams of Carbohydrate, Protein and Fat food exchanges of each of the six basic food groups (vegetable, bread, meat, skim milk, fruit and fat) for a specific individual. Sex, Age, Height, Present Weight, Frame Size, Activity Level and Basal Metabolic Rate for normal individual are taken into account. Ideal weight and sustaining calories for any weight of the above individual are calculated. Provides number of days and daily calendar after weight goal and calorie plan is determined.

F - $99.95, U - $89.95

VISA MasterCard

** SHIPPING **
Add 2% U.S.A.
(min. $2.50)
Add 5% Surface Foreign
10% Air Foreign

SOUTH EAST MEDIA
5900 Cassandra Smith Rd.
Hixson, TN 37343
info (615) 842-4601
CoCo OS-9™ FLEX™
SOFTWARE

Availability Legends —
F = FLEX, CCF = Color Computer FLEX
O = OS-9, CCO = Color Computer OS-9
U = UniFLEX
CCD = Color Computer Disk
CCT = Color Computer Tape

™FLEX is a trademark of Technical Systems Consultants
™OS9 is a trademark of Microware

!!! Please Specify Your Operating System & Disk Size !!!

TABLE 4.5

soln.	up	select	U_{RD}	U_{WR}	data bus, and interface specs.	
#	type	logic			due to U_{RD}	due to U_{WR}
2	6809E	C	'373	'373	Read data latched: up requires read data on the local bus a min.of t_{DHR} after "E" fall edge=V_L. Peripherals are req'd to hold valid read data on the global data bus a min.of t_0 after the fall edge of "E"=V_L; which is when the latch latch-es the read data.	Write data latched: up holds valid write data on the local bus a min. of t_{DHW} after "E" fall edge=V_L. Peripherals are req'd to store/latch up write data within t_{AH} +t_{UWR3} after "E" fall edge=V_L.

The combinations of read, and write buffer/latch select logic '244,'373 and '373,'244 have the same respective parameters previously described.

TABLE 4.4

soln.	up	select	U_{RD}	U_{WR}	data bus, and interface specs.	
#	type	logic			due to U_{RD}	due to U_{WR}
2	6809E	C	'244	'244	Read data real time: up requires read data on the local bus a min.of t_{DHR} after "E" fall edge=V_L. Peripherals are req'd to hold valid read data on the global data bus a min.of t_{DHR} after the fall edge of "E"=V_L. Select logic is req'd to insure up the read data specs. to be obeyed such that t_{AVS}-t_{AO}+t_{UC11}+t_{URD1} in-sures the processors read cycle require-ments.	Write data real time: up holds valid write data on the local bus a min. of t_{DHW} after "E" fall edge=V_L. Peripherals are req'd to store/latch up write data within t_{DHW} after "E" fall edge=V_L. Select logic is req'd to insure the read data buffer to be disabled before "E" rise=V_H by insuring t_{URD}+t_{UC11} to be less than t_{AO} to avoid data bus contention with a possible following write cycle next in sequence.

Notes: All special symbols are in either Signetics Corporation, or Motorola Semiconductor Products Inc. notation.

Where minimum propagation delays are req'd; if the specification is not given in data sheets implemented; assume t_0 as the minimum.

V_L (logic 0) for inputs;less than or equal to 0.7v.
(logic 0) for outputs;less than or equal to 0.7v.
(logic 0) for up inputs;less than or equal to 0.4v.

V_H (logic 1) for inputs;greater than or equal to 2.4v.
(logic 1) for outputs;greater than or equal to 2.4v.
(logic 1) for up inputs;greater than or equal to V_{CC}-0.6v.

The '373 latch must have a minimum t_0 req'd hold time after latch enable "E" fall edge=V_L.

TABLE 5

microprocessor type					timing specification symbols
6800	6802	6808	6809	6809E	description and meaning in text
t_0	t_0	t_0	t_0	t_0	exactly 0 time.
t_{c2f1}	t_{c2f1}	t_{c2f1}	t_{c2f1}	t_{c2f1}	min. fall time of c2 clock from V_H to V_L.
t_{c2f2}	t_{c2f2}	t_{c2f2}	t_{c2f2}	t_{c2f2}	max. fall time of c2 clock from V_H to V_L.
			t_{AVS2}	t_{EQ12}	max. processors clock time from "E" fall edge=V_L to "Q" rise edge greater than V_L.
----	t_{Ef1}	t_{Ef1}	t_{Ef1}	t_{Ef1}	min. fall time of microprocessor "E" clock from V_H to V_L.
----	t_{Ef2}	t_{Ef2}	t_{Ef2}	t_{Ef2}	max. fall time of microprocessor "E" clock from V_H to V_L.
----	----	----	t_{Qr1}	t_{Qr1}	min. "Q" clock rise time from V_L to V_H.
----	----	----	t_{AVS1}	t_{EQ11}	min. Processor clock time from "E" fall edge=V_L to "Q" rise edge greater than V_L.
t_{c2O1}	t_{EO1}	t_{EO1}	----	-----	min. "E", or "c2" clock low time=V_L.
t_{c2r1}	t_{Er1}	t_{Er1}	----	-----	min. "E", or "c2" clock rise time from V_L to V_H.
----	----	----	t_{Qr2}	t_{Qr2}	max. "Q" clock rise time from V_L to V_H.
t_{H1}	t_{DHR}	----	t_{DHR}	t_{DHR}	min. required processor req'd data hold. time after "c2" or "E" clock fall edge less than V_H.
t_{DSR}	t_{DSR}	t_{DSR}	t_{DSR}	t_{DSR}	min. microprocessor req'd read data setup time before "E", or "c2" fall edge less than V_H.
t_{DBEf2}	----	----	----	-----	max. fall time of "DBE" input fall edge =V_H to V_L.
t_{c2O1r}	----	----	----	-----	min. clock time low of c2=V_L till c1 rise edge greater than V_L.
----	----	----	t_{AQ}	-----	min. processor "R/W" valid time before "Q" rise greater than V_L.
t_{DBEf1}	----	----	----	-----	t_{DBEf1}=t_{c2f1}.
t_{AH}	t_{AH}	----	t_{AH}	t_{AH}	min. microprocessors guaranteed valid R/W hold time after "E", or "c2" fall edge=V_L.
----	----	----	t_{E1}	t_{E1}	min. processors "E" low time duration.
t_{c21}	----	----	----	-----	max. low time duration of "c2"=V_L from "c1"=V_H to "c2" rise edge greater than V_L.
----	----	----	t_{DHW}	t_{DHW}	min. microprocessor guaranteed valid write data hold time after "E" fall edge=V_L.
----	----	----	t_{DDQ}	t_{DDQ}	max. microprocessor guaranteed write data delay time after "Q" rise edge=V_H.
----	----	t_{U11}	t_{U11}	t_{U11}	min. U1 output disable delay to outputs tri-state.(lowest of U1 t_{PLZ}, and tPHZ time delay specifications.)
----	----	t_{U12}	t_{U12}	t_{U12}	max. U1 output disable delay to outputs tri-state.(highest of U1 t_{PLZ}, and t_{PHZ} time delay specifications.)
----	----	t_{U21}	t_{U21}	t_{U21}	min. U2 propagation delay specification from U2 input fall edge less than V_H to output rise edge less than V_H.
----	----	t_{U22}	t_{U22}	t_{U22}	max. U2 propagation delay specification from U2 input fall edge less than V_H to output=V_L.
----	----	t_{U13}	t_{U13}	t_{U13}	min. U1 outputs invalid from U1 de-select. (lowest of t_{PLZ}, and t_{PHZ} from CE rise edge greater than V_L.
----	----	----	t_{U14}	t_{U14}	min. req'd U1 valid read data setup time before U1 output enable.
----	----	----	t_{U15}	t_{U15}	max. U1 output enable delay.(max. of t_{PZL}, and t_{PZH} delay specifications.)
	t_{UA21}=t_{UB21}				min. enable propagation delay of U_{A2},or U_{B2} to outputs valid.
	t_{UA22}				max. disable propagation delay of U_{A2} to outputs tri-state.
	t_{UB22}				max. propagation delay from U_{B2} input to output.
	t_{URD1}				min. able propagation delay of U_{RD} from input to output.(lowest of U_{RD} min. t_{PZL}, and t_{PZH} delay specifications.)
	t_{URD2}				max. disable propagation delay of U_{RD} to outputs tri-state.(highest of U_{RD} min. t_{PLZ}, and t_{PHZ} delay specifications.)
	t_{UB41}				min. U_{B4} propagation delay from input=V_L to output=V_L.(min. t_{PHL} time delay specification.)
	t_{UB42}				max. U_{B4} propagation delay from input=V_L to output=V_L.(max. t_{PHL} time delay specification.)
	t_{UA11}				min. U_{A1} propagation delay from input fall edge=V_L to output fall edge less than V_L. (min. t_{PHL} time delay specification.)
	t_{URD3}				min. U_{RD} output disable propagation delay to outputs tri-state.(lowest of U_{RD}, t_{PLZ}, and t_{PHZ} time delay specifications.)
	t_{UC11}				min. U_{C1} propagation delay specification from input fall edge=V_L to output rise edge =V_L.(min. t_{PLH} time delay specification.)
	t_{WR3}				min. U_{WR} output disable propagation delay to U_{RD} outputs tri-state.(lowest of U_{WR} t_{PLZ}, and t_{PHZ} time delay specifications.)
	t_{UB11}				min. U_{B1} propagation delay from input fall edge=V_L to output fall edge less than V_H. (min. t_{PHL} time delay specification.)
	t_{UB12}				max. U_{B1} propagation delay from input fall edge=V_L to output fall edge less than V_H. (max. t_{PHL} time delay specification.)
	t_{URD5}				min. U_{RD} propagation delay specifications. (lowest of t_{PLH}, and t_{PHL} time delay specifications.)
	t_{WR4}				min. U_{WR} propagation delay specifications. (lowest of t_{PLH}, and t_{PHL} time delay specifications.)
	t_{UC12}				max. U_{C1} propagation delay specification from input fall edge=V_L to output rise edge =V_L.(max. t_{PLH} time delay specification.)
	t_{URD4}				max. U_{RD} output enable propagation delay specification from input to valid output. (highest of t_{PZL}, and t_{PZH} output enable time.)

Bit Bucket

COBOL Name & Address System

by Mike Martin

Continued from last month.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. NAMP003B.
ENVIRONMENT DIVISON.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT NAME-FILE ASSIGN TO "WNAMD0001DAT"
        ACCESS MODE IS RANDOM
        FILE STATUS IS NAME-STAT.
DATA DIVISON.
FILE SECTION.
PD  NAME-FILE.
01  NAME-RECORD PIC X(126).
WORKING-STORAGE SECTION.
     COPY "WNAMCDINQCBL".
    COPY "WWFCCDT00CBL".
    COPY "WNAMCDMSTCBL".
PROCEDURE DIVISION.
C100-CLEAR-FIELDS SECTION.
    MOVE "1" TO OV-SWITCH.
    MOVE 5 TO X.
    MOVE 13 TO Y.
    PERFORM Z110-ACA THRU Z110X.
    PERFORM C200-DISPLAY-DASH THRU C200X 20 TIMES.
    MOVE 45 TO Y.
    PERFORM Z110-ACA THRU Z110X.
    PERFORM C200-DISPLAY-DASH THRU C200X 15 TIMES.
    MOVE 7 TO X.
    MOVE 13 TO Y.
    PERFORM Z110-ACA THRU Z110X.
    PERFORM C200-DISPLAY-DASH THRU C200X 20 TIMES.
    MOVE 45 TO Y.
    PERFORM Z110-ACA THRU Z110X.
    PERFORM C200-DISPLAY-DASH THRU C200X 20 TIMES.
    MOVE 9 TO X.
    MOVE 13 TO Y.
    PERFORM Z110-ACA THRU Z110X.
    PERFORM C200-DISPLAY-DASH THRU C200X 10 TIMES.
    MOVE 31 TO Y.
    PERFORM Z110-ACA THRU Z110X.
    PERFORM C200-DISPLAY-DASH THRU C200X 2 TIMES.
    MOVE 39 TO Y.
    PERFORM Z110-ACA THRU Z110X.
    PERFORM C200-DISPLAY-DASH THRU C200X 9 TIMES.
    MOVE 54 TO Y.
    PERFORM Z110-ACA THRU Z110X.
    PERFORM C200-OISPLAY-OASH THRU C200X 10 TIMES.
    MOVE 12 TO Y.
    MOVE 11 TO X.
    PERFORM C300-CLEAR-FLAGS THRU C300X 4 TIMES.
    MOVE 16 TO X.
    MOVE 1 TO T.
    PERFORM Z110-ACA THRU Z110X.
    PERFORM Z020-EOL-ERASE THRU Z020X.
    MOVE 5 TO X.
    MOVE 45 TO Y.
    PERFORM Z110-ACA THRU Z110X.
    CALL "NAMP003A" OVERLAY.
C100X. EXIT.
C200-DISPLAY-DASH.
    DISPLAY I-0 "_" UPON CONSOLE.
C200X. EXIT.
C300-CLEAR-FLAGS.
    PERFORM Z110-ACA THRU Z110X.
    PERFORM C200-DISPLAY-DASH THRU C200X.
    ADO 1 TO X.
C300X. EXIT.
Z000-TERMINAL-CONTROL SECTION.
```

```
Z001-OELAY.
    ADD 1 TO DELAY-VAR.
Z001X. EXIT.
Z020-EOL-ERASE.
    DISPLAY I-0 SOS UPON CONSOLE.
    MOVE ZEROES TO DELAY-VAR.
    PERFORM Z001-DELAY THRU Z001X
        UNTIL DELAY-VAR > 600.
Z020X. EXIT.
Z110-ACA.
    DISPLAY I-0 $1B $3D
        ACA-ADDRESS (X) ACA-ADDRESS (Y)
        UPON CONSOLE.
Z110X. EXIT.


IDENTIFICATION DIVISION.
PROGRAM-ID. NAMP0004.
ENVIRONMENT OIVISON.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT NAME-FILE ASSIGN TO "WNAMD00010AT"
        ACCESS IS RANDOM
        FILE STATUS IS NAME-STAT.
DATA OIVISON.
FILE SECTION.
FO  NAME-FILE.
01  NAME-RECORD PIC X(126).
WORKING-STORAGE SECTION.
    COPY "WNAMCDUPDCBL".
    COPY "WWFCCDT00CBL".
    COPY "WNAMCDMSTCBL".
PROCEDURE DIVISION.
    COPY "WWFCCPT01CBL".
    MOVE 1 TO Y.
    MOVE 5 TO X.
    PERFORM Z110-ACA THRU Z110X.
    DISPLAY I-0 "FIRST NAME:"
        UPON CONSOLE.
    MOVE 34 TO Y.
    PERFORM Z110-ACA THRU Z110X.
    DISPLAY I-0 "LAST NAME:"
        UPON CONSOLE.
    MOVE 7 TO X.
    MOVE 1 TO Y.
    PERFORM Z110-ACA THRU Z110X.
    DISPLAY I-0 "ADDRESS 1:"
        UPON CONSOLE.
    MOVE 34 TO Y.
    PERFORM Z110-ACA THRU Z110X.
    DISPLAY I-0 "ADDRESS 2:"
        UPON CONSOLE.
    MOVE 9 TO X.
    MOVE 1 TO Y.
    PERFORM Z110-ACA THRU Z110X.
    DISPLAY I-0 "CITY:" UPON CONSOLE.
    MOVE 24 TO Y.
    PERFORM Z110-ACA THRU Z110X.
    DISPLAY I-0 "STATE:" UPON CONSOLE.
    MOVE 34 TO Y.
    PERFORM Z110-ACA THRU Z110X.
    DISPLAY I-0 "ZIP:" UPON CONSOLE.
    MOVE 49 TO Y.
    PERFORM Z110-ACA THRU Z110X.
    DISPLAY I-0 "TEL:" UPON CONSOLE.
    MOVE 11 TO X.
    MOVE 2 TO Y.
    PERFORM Z110-ACA THRU Z110X.
    DISPLAY I-0 "XMAS:" UPON CONSOLE.
    MOVE 12 TO X.
    PERFORM Z110-ACA THRU Z110X.
    DISPLAY I-0 "PARTY:" UPON CONSOLE.
    MOVE 13 TO X.
    PERFORM Z110-ACA THRU Z110X.
    DISPLAY I-0 "BUSINESS:" UPON CONSOLE.
    MOVE 14 TO X.
    PERFORM Z110-ACA THRU Z110X.
    DISPLAY I-0 "BILL:" UPON CONSOLE.
    OPEN I-0 NAME-FILE.
    CALL "NAMP004B" OVERLAY.
    COPY "WWFCCPT00CBL".
```

```
IDENTIFICATION DIVISION.
PROGRAM-ID. NAMP004A.
ENVIRONMENT DIVISON.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT MAME-FILE ASSIGN TO "WNAMD0001DAT"
        ACCESS MODE IS RANDOM
        FILE STATUS IS NAME-STAT.
DATA DIVISON.
FILE SECTION.
FO  NAME-FILE.
01  MAME-RECORD PIC X(126).
WORKING-STORAGE SECTION.
    COPY "WNAMCDUPDCBL".
    COPY "WWPCCDTOOCBL".
    COPY "WNAMCONSTCBL".
PROCEDURE DIVISION.
A000-UPDATE-CONTROL SECTION.
    ACCEPT WS-LAST-NAME FROM CONSOLE.
    IF WS-LAST-NAME = "END"
        CLOSE NAME-FILE
        CALL "NAMP0001" CHAIN.
    MOVE ZERO TO WS-REC-NO.
    PERFORM A100-UPDATE-LOOP THRU A100X
        UNTIL WS-NAR-LAST-NAME = "!".
    MOVE SPACES TO WS-MAME-ADDRESS-RECORD.
    CALL "NAMP004B" OVERLAY.
A000X. EXIT.
A100-UPDATE-LOOP SECTION.
    MOVE 0 TO EOF-FLAG.
    PERFORM R100-READ THRU R100X
        VARYING WS-REC-NO FROM WS-REC-NO BY 1
          UNTIL WS-NAR-LAST-NAME = WS-LAST-NAME.
    IF EOF-FLAG = 1
        GO TO A100X.
    PERFORM B000-DISPLAY THRU B000X.
A101-ASK.
    MOVE 16 TO X.
    MOVE 1 TO Y.
    PERFORM Z110-ACA THRU Z110X.
    PERFORM Z020-EOL-ERASE THRU Z020X.
    DISPLAY 1-O "Enter ´u´-update ´D´-delete ´C´ continue "
        UPON CONSOLE.
    ACCEPT WS-RESPONSE FROM CONSOLE.
    IF WS-RESPONSE = "C"
        MOVE SPACES TO WS-NAR-LAST-NAME
        GO TO A100X.
    IF WS-RESPONSE = "D"
        MOVE ALL SPACES TO WS-NAME-ADDRESS-RECORD
        MOVE "@" TO WS-MAR-EOR
    ELSE
        IF WS-RESPONSE = "U"
            PERFORM B100-NEW-DATA THRU B199X
        ELSE
            GO TO A101-ASK.
    WRITE MAME-RECORD FROM WS-NAME-ADDRESS-RECORD.
    MOVE 16 TO X.
    MOVE 1 TO Y.
    PERFORM Z110-ACA THRU Z110X.
    PERFORM Z020-EOL-ERASE THRU Z020X.
    DISPLAY 1-O "Enter ´S´ to continue search, ´E´ to end "
        UPON CONSOLE.
    ACCEPT WS-RESPONSE FROM CONSOLE.
    IF WS-RESPONSE = "E"
        MOVE 1 TO EOF-FLAG
        MOVE "!" TO WS-MAR-LAST-NAME
        GO TO A100X.
    ADD 1 TO WS-REC-NO.
A100X. EXIT.
B000-DISPLAY SECTION.
    MOVE 5 TO X.
    MOVE 13 TO Y.
    PERFORM Z110-ACA THRU Z110X.
    DISPLAY I-O WS-MAR-FIRST-NAME UPON CONSOLE.
    MOVE 45 TO Y.
    PERFORM Z110-ACA THRU Z110X.
    DISPLAY 1-O WS-MAR-LAST-NAME UPON CONSOLE.
    MOVE 7 TO X.
    MOVE 13 TO Y.
    PERFORM Z110-ACA THRU Z110X.
    DISPLAY I-O WS-MAR-LINE1 UPON CONSOLE.
    MOVE 45 TO Y.
    PERFORM Z110-ACA THRU Z110X.
    DISPLAY 1-O WS-NAR-LINE2 UPON CONSOLE.
    MOVE 9 TO X.
    MOVE 13 TO Y.
    PERFORM Z110-ACA THRU Z110X.
    DISPLAY 1-O WS-NAR-CITY UPON CONSOLE.
    MOVE 31 TO Y.
    PERFORM Z110-ACA THRU Z110X.
    DISPLAY I-O WS-NAR-STATE UPON CONSOLE.
    MOVE 39 TO Y.
    PERFORM Z110-ACA THRU Z110X.
    DISPLAY I-O WS-NAR-ZIP UPON CONSOLE.
    MOVE 54 TO Y.
    PERFORM Z110-ACA THRU Z110X.
    DISPLAY I-O WS-NAR-PHONE UPON CONSOLE.
    MOVE 11 TO X.
    MOVE 12 TO Y.
    PERFORM Z110-ACA THRU Z110X.
    DISPLAY I-O WS-NAR-XMAS UPON CONSOLE.
    MOVE 12 TO X.
    PERFORM Z110-ACA THRU Z110X.
    DISPLAY I-O WS-NAR-PARTY UPON CONSOLE.
    MOVE 13 TO X.
    PERFORM Z110-ACA THRU Z110X.
    DISPLAY I-O WS-NAR-BUSINESS UPON CONSOLE.
    MOVE 14 TO X.
    PERFORM Z110-ACA THRU Z110X.
    DISPLAY I-O WS-NAR-BILL UPON CONSOLE.
B000X. EXIT.
```

Continued next month...

The SIZE Utility

(w/ a FLEX Command Line Parsing Module)

Mark Armstrong
22 Alexander Ave,
Torbay,
Auckland 10.
NEW ZEALAND.
ph (09) 403 7924

Dear Don,

Enclosed is a 6809 FLEX disk containing the source and object of a program for publication in 68' Micro Journal. The source files to be printed are "SIZE.TXT" and "PARSE.TXT" and "SIZE.DOC" contains the documentation for the program.

This started as an upgrade of my SIZE program so that it would give me the true size of binary files by taking any overlap in the load map into account. Then I thought it would be nice to make the default filename extension CMD as typically I use SIZE on command files. Well, by the time the snowball stopped rolling, I had my upgrade and a general purpose parse module which eases the task of parsing options and numbers from the command line as well as file specifications.

I am presenting the complete source of SIZE (includes the source of the PARSE module) so that other readers do not have to re-invent the wheel and as an example of but one method of handling command line parsing - something which occurs all the time when writing programs under FLEX.

The PARSE Module

This is a self-contained module which is separately compiled and the compiled code is just linked in when compiling SIZE. If you want a complete source listing when compiling SIZE, replace the line in SIZE

```
module ParseModule=code from "PARSE";
```

with the line

```
~ include "PARSE"
```

If you want SIZE to be a single source file, replace the line with the entire text of PARSE.

The variables and procedures in the module are documented in the source and how you use them is demonstrated in SIZE. However, a few notes may be useful;

1) NextCh uses NXTCH in FLEX so there are no worries about it crossing into the next command on the command line.

2) To parse say an integer on the command line, do this;

   ```
   ReturnCurrCh:=true;
   read from NextCh TheInteger;
   ```

   CurrCh will be left containing the character immediately after the integer. WHIMSICAL's read integer will skip any leading spaces.

3) All programs should keep parsing the command line (i.e. calling NextCh) until they hit either a TTYSET End-Of-Line character or a Carriage Return so that FLEX will recognise any remaining commands on the command line. This is what FinishParse is intended for. It should be used whenever the program aborts while parsing the command line and when the program has all the parameters it needs fom the command line. Note that if the program is aborted via the PCRLF system routine, FLEX will ignore the remainder of the command line.

   Unfortunately, some programs (even commercial ones) do not always do the above and some even use the line buffer for their own use, destroying the command line and any unprocessed commands on it.

The SIZE Program

I guess the first thing to notice is that both uppercase and lowercase M are accepted as the Map option. All too often programs prompt us with a question to which we answer "y" and it is taken as a no!.

The ParseOptions procedure really has unnecessary features. The whole procedure could be replaced by

```
  proc ParseOptions=
% ---- ------------
  begin
    NextCh; case CurrCh of {"M":"m": LoadMap:=true};
  end;
```

However, I wanted to present a general purpose structure that is easily expanded for more options. These can simply be added to the case statement as required.

Note the main "while loop" skips any nulls between records. Nulls can be found in the middle of a file formed by an APPEND operation and will usually be encountered at the end of a file (padding out the last sector).

Ndx=0 is used to indicate that the loop has just been entered. Also note that the addresses in the Last array are actually one bigger than the last address. This is why the loadmap is written as Last[Ndx]-$0001 but it saves having to add or subtract $0001 at various other places in the program.

The variable Bytes is a LARGEINT (a four byte integer) because this is the most straightforward method for getting SIZE to report the size of files bigger than $7FFF correctly.

When reading a DBYTE (such as Transfer) from the BYTE FILE Binary, WHIMSICAL reads two consecutive bytes from the file and automatically combines then into the DBYTE - a handy feature.

The curly braces { and } are equivilent to a begin end pair and are used just for style. The vertical bar "|" character is equivilent to a space and has been used extensively in SIZE and PARSE to give a visual guide to the structure of the program wherever the indenting made it not obvious.

Conclusion

I was amazed how long it was since I last contributed to 68' Micro Journal. I've even moved to a new address since then. I guess I was too busy reading all the interesting material others have contributed. I look forward to each issue and think you're all doing a fine job. Keep up the good work.

Regards,

Mark Armstrong

(Mark Armstrong)

```
% Determine Size Of Binary Files (File SIZE) 2 JUN 85
% --------- ---- -- ------ -----

% by M G Armstrong.
% 22 Alexander Ave,
% Torbay,
% Auckland 10.
% NEW ZEALAND.
% ph (09) 403 7924

% compiled by WHIM VER 1.7:88(4)

% Command Syntax:
%    SIZE,<file spec>[,+<option>]
% where file spec defaults to work drive with CMD extension
% and the only option is M to display the file load map.

- STACK=([$CC28])
- VERSION 3,".SIZE by M G Armstrong in WHIMSICAL"
begin
  BOOL        LoadMap,            % write the file load map if true
              TransferFound;      % true if a transfer address found
  BYTE        Data,               % used for binary record data
              Count;              % number of data bytes in record
  DBYTE       Transfer,           % last encountered transfer address
              Address;            % record load address
  LARGEINT    Bytes;              % number of unique addresses occupied
  SMALLINT    I,                  % general purpose array index
              Ndx;                % index of last load map entry
  DBYTE ARRAY First[127],         % first address of consecutive records
              Last[127];          % last address of consecutive records+$0001
  CHAR ARRAY  Name[14],           % file name
              CMD=("CMD");        % default extension
  BYTE FILE   Binary;             % the binary file

  proc PCRLF=external($CD24);     % print CR, LF and look for ESC

module ParseModule=code from "PARSE";

  proc ParseOptions=
% ---- ------------
% Parse options from the command line. Embedded spaces are allowed.
  begin
    BOOL Quit;
    do begin
    | NextCh; % get next option
    | case CurrCh of
    | begin
    |   " ":        ;
    |   "M":"m": LoadMap:=true;
    |   else:    Quit:=true;
    | end;
    end until Quit;
  end;

  proc RemoveOverlap(SMALLINT N)=
% ---- -------------
% Ensure that map address entry N does not overlap any other entries.
  begin
    SMALLINT I;
```

```
    for I:=1 to Ndx do % scan all entries
    if I<>N then       % skipping entry N
    begin
      if First[N]>=First[I] AND First[N]<Last[I] then % overlap
      if Last[N]<=Last[I] then
      begin
        Last[N]:=First[N]; % if subset then nullify
      end else
      begin
        First[N]:=Last[I]; % remove the overlap
      end;
    end;
  end;
% Main program starts here
% ---- ------- ------ ----
  PCRLF;
  if ParseFileName(Name, CMD) then
  begin
    write "Illegal file specification";
    FinishParse;                 % ensure that parse is completed
    STOP;                        % return to FLEX immediately
  end;
  if CurrCh="+" then ParseOptions else
  if NextCh="+" then ParseOptions; % check for options
  FinishParse;                     % ensure that parse is completed

  open Binary as Name;           % open the file for read
  read from Binary Data;         % get the first byte
  if Data<>$02 AND Data<>$16 then  % check if start of record indicator
  begin                            % or transfer address indicator
    write "Not a binary file";
    STOP;                        % return to FLEX immediately
  end;

  % if it is a start of record or transfer address indicator, OR it is a
  % null (e.g. resulting from an APPEND operation) AND not the
  % end of the file then keep going
  while (Data=$02 OR Data=$16) OR (Data=$00 AND NOT EOF(Binary)) do
  begin
    if Data=$02 then
    begin
      read from Binary Address, Count; % read load address and byte count
      if Ndx=0 OR (Address<>Last[Ndx]) then
      begin
        if Ndx=127 then
        begin
          write "^C^M^JLoad map too fragmented";
          STOP;
        end;
        Ndx:=Ndx+1;
        First[Ndx]:=Address;
        Last[Ndx]:=Address+COMBINE($00, Count);
        if LoadMap then
        begin
          if Ndx>1 then
          begin
            write "$", Last[Ndx-1]-$0001;
            PCRLF;
          end;
          write "Loads from $", First[Ndx], " to ";
        end;
      end else
      begin
        Last[Ndx]:=Address+COMBINE($00, Count);
      end;
      while Count>$00 do {read from Binary Data; Count:=Count-$01};
    end else
    if Data=$16 then
    begin
      read from Binary Transfer;
      TransferFound:=true;
    end;
    if NOT EOF(Binary) then read from Binary Data;
  end;

  if Ndx>0 then % something was found
  begin
    if LoadMap then
    begin
      write "$", Last[Ndx]-$0001;
      PCRLF;
      if TransferFound then
      begin
        write "Transfer address is $", Transfer;
        PCRLF;
      end;
      PCRLF;
    end;
    for I:=1 to Ndx do RemoveOverlap(I);
    for I:=1 to Ndx do Bytes:=Bytes+EXTEND(DEC(Last[I]-First[I]));
  end;
  write "File is ", Bytes, " bytes long";
end.

% -----------------------------------------------------------------
```

```
% General Purpose Parse Module (File PARSE) 2 JUN 85
% ------- ------- ----- ------

% by M C Armstrong.
% 22 Alexander Ave,
% Torbay,
% Auckland 10.
% NEW ZEALAND.
% ph (09) 403 7924

% compiled by WHIM VER 1.7:88(4)

module ParseModule=
begin
public
  CHAR CurrCh($CC18);
% ---- ------
% This is the current character on the command line.

  BOOL ReturnCurrCh;
% ---- ------------
% If true, NextCh will return CurrCh when next called.  NextCh  always
% sets  ReturnCurrCh  false.  This  allows the use of WHIMSICAL's read
% <number> starting with CurrCh and leaving CurrCh as the terminator.

  CHAR proc NextCh;
% ---- ---- ------
% This returns the next character on the command line.
% Multiple spaces are skipped and returned as a single space.

  BOOL proc FinishParse;
% ---- ---- -----------
% Calls NextCh until CurrCh is either the TTYSET End Of Line character
% or a Carriage Return. Returns true if remainder of  line  was  other
% than spaces.

  BOOL proc ParseFileName(CHAR ARRAY Name, DefaultExtension);
% ---- ---- -------------
% Parses  the  file  spec on the command line into Name, starting with
% CurrCh. Returns true if there is a format error in  the  file  spec.
% Name  must  be  declared  with  at least 15 elements and the first 3
% elements of DefaultExtension are used as the default extension.

private
  CHAR proc NextCh=
% ---- ---- ------
  begin
    DBYTE proc NXTCH=external($CD27);
    if ReturnCurrCh then
    begin
      NextCh:=CurrCh;
      ReturnCurrCh:=false;
    end else
    begin
      NextCh:=CHR(HIBYTE(NXTCH));
    end;
  end;

  BOOL proc FinishParse=
% ---- ---- -----------
  begin
    CHAR TTYEOL($CC02);
    while CurrCh<>TTYEOL AND CurrCh<>CHR($00) do
    begin
      if CurrCh<>" " then FinishParse:=true;
      NextCh;
    end;
  end;
  BOOL proc GetFileName(CHAR ARRAY Name)=code
% ---- ---- -----------
% Returns true if there is a format error in the file spec.
  ( $BD, $C02D,       % JSR  GETFIL    address of Name passed in (X)
    $24, $01,         % BCC  CLRB      no errors so clear (B)
    $C6, $5F,         % LDB  #CLRB     errors, so make (B) non zero
    $39 );            % RTS            return

  BOOL proc ParseFileName(CHAR ARRAY Name, DefaultExtension)=
% ---- ---- -------------
% Returns true if there is a format error in the file spec.
  begin
    SMALLINT Source,
             Dest:=1; % note declaration time assignment
    ParseFileName:=GetFileName(Name);
    Name[0]:=CHR(ASC(Name[3])+$30); % convert binary to ASCII
    for Source:=4 to 14 do
    begin
      case Source of % put in the dots
      begin
        4:12: Name[Dest]:=".";
              Dest:=Dest+1;
      end;
      if Name[Source]<>CHR($00) then % do not transfer nulls
      begin
        Name[Dest]:=Name[Source];
        Dest:=Dest+1;
      end else
      case Source of % if extension null then put in default
      begin
        12,14: Name[Dest]:=DefaultExtension[Source-12];
               Dest:=Dest+1;
      end;
    end;
    Name[Dest]:=CHR($00); % terminate file specification with a null
  end;                    % for compatability with STK modules

end; % of parse module

% -----------------------------------------------------------------
```

SIZE

The SIZE command is used to determine the size of a binary format file. The size is given in decimal and is the number of unique addresses the file loads into. An option is provided for displaying the file load map.

DESCRIPTION

The general syntax of the SIZE command is:

```
SIZE,<file spec>[,+<option>]
```

where the file spec defaults to the work drive with CMD extension and the only option is;

M Display the load map. The first address and last address of each block of consecutive binary data is displayed. If a transfer address is found in the file, it follows the list of addresses. If more than one transfer address is found, only the last one is retained and displayed since it is the one FLEX will use.

Examples:

```
+++SIZE MAKE +++SIZE OVERLAY.BIN+M
+++SIZE 0.VIEW+M
```

The first example would report the size of MAKE.CMD. The second example would show where OVERLAY.BIN loads and its size. The last example would show where 0.VIEW.CMD loads and its size.

NOTE: These programs are available on **68 Micro Journal Reader Service Disk** #18 (see p. 60).

The CMD Utility

John Spray
8 Valley View Rd.
Glenfield,
Auckland 10.
NEW ZEALAND.
ph (09) 444 6550

Dear Don,

Please find on the enclosed disk the source and object of a program for publication in 68' Micro Journal. The source files to be printed are "CMD.TXT" and "CMDCODE.TXT" and "CMD.DOC" contains the documentation for the program. CMDCODE is part of the CMD program and should be published with it. The program is included with this disk from Mark Armstrong because it makes use of his module "PARSE" also.

Your faithfully,

John Spray

This is a neat little utility which offers an alternative to using an EXEC file to avoid typing a frequently used command line. For example I often have to set my printer to emphasised mode by using the command sequence:

```
P4 HECHO 1B 45 07
```

The final bell, by the way, is only to make sure the printer receives the command. I never actually type in that command line. Instead I simply type EMPH<cr>. This utility will load the above command string into the FLEX command buffer and re-enter FLEX to have it executed.

The CMD utility allows simple creation of such utilities for any desired command string. To create EMPH.CMD you simply type:

```
CMD,EMPH,P4 HECHO 1B 45 07
```

The syntax for the CMD utility is as follows:

```
CMD,filename,command-string
```

The filename is the name of the CMD file to be created. Of course the default extension is .CMD. It is created on the system drive unless specified otherwise. The command-string is any valid FLEX command line which you wish to use often. It may contain multiple commands separated by TTYSET End of Line characters (usually a colon). The entire command line will stored into the new utility.

Utilities created by CMD will shift the tail of the command line along if necessary before copying the command string in. Therefore commands following will still execute correctly. For example if you type:

```
EMPH:P4 LIST PROGRAM
```

The EMPH utility will create this string in the buffer before returning to the FLEX RENTER point.

```
P4 HECHO 1B 45 07:P4 LIST PROGRAM
```

Say you wanted to create a new utility to perform most of the above command, so that you could type EMPRINT PROGRAM<cr> to have a program printed in emphasised print mode. That is everything up to and including LIST. You would type:

```
CMD,EMPRINT,P4 HECHO 1B 45 07|P4 LIST
```

Now say you wanted to have an option +P on the end as part of the standard EMPRINT utility. In this case the program name must be inserted into the middle of the command string as a parameter. To do that you use an "@" character with a parameter number, for example:

```
CMD,EMPRINT,P4 HECHO 1B 45 07|P4 LIST
@1 +P
```

Now when you type EMPRINT PROGRAM<cr> the string PROGRAM is substituted into the command string where the @1 was.

Up to 9 parameters may be specified. The parameters are taken in order from the command line and substituted wherever the corresponding parameter numbers are in the string. Parameters are separated by spaces or commas unless enclosed by matching quotes, either " or ´. If enclosed by quotes any characters may be passed including ttyset end of line characters. If fewer parameters are passed than required by a given command, the null strings are used for the missing parameters. If a null parameter is required it can always be specified by using two successive commas.

The same parameter number may occur several times in the string. The substitute character, the "@", can be changed to any of $ 2 & * or ~ by a specifying it as an option after the filename, for example:

```
CMD COMPILE +$ PASCAL $1 +YCQ$2:ASMB
$1.TXT +YSL$3:PRUN $1
```

The above command will create the utility COMPILE which will compile, assemble and run a pascal program.

Parameters add great flexibility to the CMD utility. The only catch is that the resulting command string must fit into the FLEX command buffer.

The utilities created by the CMD utility have one extra little feature. They echo to the screen the entire command string just before the return to FLEX so that you can see the exact commands which will be executed as if you had typed them yourself. This feature can be suppressed by using a +S option when the CMD utility is used.

```
% Utility for making a command file to execute a
% specific command

% Written by J.R.SPRAT in Whimsical
```

```
~STACK=[$CC28]
~VERSION 2

BEGIN

  CHAR ARRAY NAME[14],EXTENSION=("CMD");% File name and extension
  BYTE STRINGLEN,                               % Length of string
       SYSTEMDRIVE($CC0B),
       WORKDRIVESAVE,
       WORKDRIVE($CC0C);
  CHAR TTYEOL($CC02),                       % Flex's ttyset eol character
       TTYEOLSAVE;                          % Save area for ttyeol
  BOOL PARAM;
  DBYTE SPTR,                               % Pointer into CMDCODE
        PATCHES;                            % Loc for patches in CMDCODE

  % Following procedure is not a procedure at all but a way of
  % getting the CMDCODE program in
  PROCEDURE PCMDCODE=CODE FROM "CMDCODE.CMD";
  PROCEDURE DUMMY=CODE(); % This so we know where end of pcmdcode is

  % Following array overlays the PCMDCODE procedure
  BYTE ARRAY CMDCODE(LOC(PCMDCODE));

  MODULE PARSE=CODE FROM "PARSE";           % Module to parse filename
  PROCEDURR PCRLF=EXTERNAL($C024);

  PROCEDURE FLUSH=                          % Procedure to emit code
  BEGIN
      BYTE FILE CMDFILE;
      DBYTE TRANSFER=$C100,                 % Transfer address
            CODAD:=TRANSFER,                % Current emit code address
            SPTR,                           % Pointer into CMDCODE
            LENGTH;                         % Length of CMDCODE

      PROC WRITEREC(BYTE COUNT);
      BEGIN
        WRITE TO CMDFILE $02,CODAD,COUNT;
        WHILE COUNT>$00 DO
        BEGIN
          WRITE TO CMDFILE CMDCODE[SPTR];
          SPTR:=SPTR+$0001;
          CODAD:=CODAD+$0001;
          COUNT:=COUNT-$01;
        END;
      END;

    CREATE CMDFILE AS NAME;                   % Create Flex binary file
    LENGTH:=LOC(DUMMY)-LOC(PCMDCODE);
    WHILE SPTR<LENGTH DO
    BEGIN
      IF LENGTH-SPTR>$00FF THEN WRITEREC($FF)
      ELSE WRITEREC(LOBYTE(LENGTH-SPTR));
    END;
    WRITE TO CMDFILE $16,TRANSFER; % Write out transfer address
    CLOSE CMDFILE;
  END;    % OF FLUSH PROCEDURE
% MAIN PROGRAM STARTS HERE
PCRLF;
WORKDRIVESAVE:=WORKDRIVE;
WORKDRIVE:=SYSTEMDRIVE;                   % Set working drive to system
IF PARSEFILENAME(NAME,EXTENSION) THEN
BEGIN
  WRITE "Invalid file specification";
  WORKDRIVE:=WORKDRIVESAVE;
  STOP;
END;
WORKDRIVE:=WORKDRIVESAVE;
IF CURRCH=" " OR CURRCH="," THEN NEXTCH;

% Find the place to put the patches
WHILE CMDCODE[PATCHES]<>$00 OR CMDCODE[PATCHES+$0001]<>$FF DO
  PATCHES:=PATCHES+$0001;

% Parse any options
IF CURRCH='+' THEN % options
BEGIN
  NEXTCH;
  DO
    CASE CURRCH OF
    BEGIN
      'S': CMDCODE[PATCHES+$0001]:=$00; % Suppress echo
      '$','&': '*': '~': % Alternative substitute chars
        CMDCODE[PATCHES+$0002]:=ASC(CURRCH);
      ELSE: WRITE "Invalid option"; STOP;
    END;
    NEXTCH;
  UNTIL CURRCH=' ' OR CURRCH=',';
  NEXTCH;
END;


% Get the string including ttyeol characters
TTYEOLSAVE:=TTYEOL;
TTYEDL:=CHR($00);             % Set ttyeol to null
IF CURRCH="^M" THEN           % No string given
```

```
  BEGIN
    WRITE "String expected";
    STOP;
  END;
  SPTR:=PATCHES+$0004;            % Point SPTR to string
  DO BEGIN                        % Read in text string
    CMDCODE[SPTR]:=ASC(CURRCH);
    STRINGLEN:=STRINGLEN+$01;
    SPTR:=SPTR+$0001;
    IF PARAM AND CURRCH>='1' AND CURRCH<='9' THEN
      IF CMDCODE[PATCHES+$0003]<ASC(CURRCH) THEN
        CMDCODE[PATCHES+$0003]:=ASC(CURRCH); % Set no params
    PARAM:=CURRCH=CHR(CMDCODE[PATCHES+$0002]); % Param?
    NEXTCH;                                   % Get next character
  END UNTIL CURRCH="^M";                      % Finish on cr only
  CMDCODE[PATCHES]:=STRINGLEN;                % Patch length of string
  TTYEOL:=TTYEOLSAVE;                         % Put back ttyeol

  % Create the new utility
  FLUSH;
  WRITE "Command file created.";
END.

% CMDCODE.TXT
% This utility is the shell used by cmd to create specialized
% utilities. It is incorporated as part of cmd.

% Written by J.R.SPRAY in Whimsical

^STACK=[$CC2B]        % Start stack at memend
^VERSION 2,"Created by the CMD utility ver 2"
BEGIN
  DBYTE BUFLOC=$C080;            % Flex's line buffer location
  CHAR ARRAY BUF(BUFLOC);        % Flex's line buffer
  DBYTE BUFPTR($CC14);           % Flex's line buffer pointer
  CHAR TTYEOL($CC02),            % Flex's end of line character
       LSTTRM($CC11);            % Flex's last terminator

  % Following 5 arrays are filled in by the cmd utility

  BYTE ARRAY LENGTH=($00);       % Length of string
  BOOL ARRAY ECHOFL=(TRUE);      % Echo the string flag
  CHAR ARRAY SUBSTCH=("@");      % Substitute character
  CHAR ARRAY NPARAMS=("0");      % Number of parameters
  CHAR ARRAY STRING=(            % String of 128 characters
"                                                              ",
"                                                              ",
"                                  ");

  BYTE LBP:=LOBYTE(BUFPTR-BUFLOC), % Line buffer pointer
       PL;                         % Parameter length
  CHAR PC:="1",                    % Parameter counter
       DELIM,                      % Parameter delimiter
       CURCH;                      % Current character
  BYTE SPTR,                       % Search pointer
       MPTR,                       % Move pointer
       COUNT;

  PROCEDURE PCRLF=EXTERNAL($C024);

  PROCEDURE SKIPSEP=
  BEGIN
    WHILE BUF[LBP]=' ' DO LBP:=LBP+$01;
    IF BUF[LBP]=',' THEN LBP:=LBP+$01;
    CURCH:=BUF[LBP];
  END;

  PCRLF;
  WHILE PC<=NPARAMS[0] DO % Substitute the parameters
  BEGIN
    SKIPSEP; % Skip seperators
    % Determine parameter delimiter
    DELIM:=CURCH;
    IF DELIM="'" OR DELIM='"' THEN
      LBP:=LBP+$01
    ELSE
      DELIM:=' ';
    % Find length of parameter
    PL:=LBP;
    CURCH:=BUF[PL];
    WHILE CURCH<>DELIM AND CURCH<>"^M"
          AND (CURCH<>',' AND CURCH<>TTYEOL OR DELIM<>' ') DO
    BEGIN
      PL:=PL+$01;
      CURCH:=BUF[PL];
    END;
    PL:=PL-LBP;
    SPTR:=$00; % Reset search pointer
    WHILE SPTR<LENGTH[0] DO % Search string for parameters
    BEGIN
      IF STRING[SPTR]=SUBSTCH[0] AND STRING[SPTR+$01]=PC THEN
      BEGIN
        IF LENGTH[0]+PL-$2>=$80 THEN
        BEGIN
          WRITE "Insufficient room to substitute parameters";
          STOP;
        END;
        IF PL>$2 THEN
        BEGIN
          MPTR:=LENGTH[0];
          WHILE MPTR>SPTR DO
          BEGIN
            MPTR:=MPTR-$01;
            STRING[MPTR+PL-$2]:=STRING[MPTR];
          END;
        END ELSE
        BEGIN
          MPTR:=SPTR;
          WHILE MPTR<LENGTH[0] DO
          BEGIN
            STRING[MPTR]:=STRING[MPTR+$2-PL];
            MPTR:=MPTR+$01;
          END;
        END;
        LENGTH[0]:=LENGTH[0]+PL-$02;
        MPTR:=LBP;        % Point MPTR to the parameter
        COUNT:=PL;        % Set count to parameter length
        WHILE COUNT>$00 DO % Transfer the parameter
        BEGIN
          STRING[SPTR]:=BUF[MPTR];
          MPTR:=MPTR+$01;
          SPTR:=SPTR+$01;
          COUNT:=COUNT-$01;
        END;
      END ELSE SPTR:=SPTR+$01;
    END;
    LBP:=LBP+PL; % Skip past parameter
    IF DELIM<>' ' AND DELIM=CURCH THEN % Skip ending quote
      LBP:=LBP+$01;
    PC:=CHR(ASC(PC)+$01);
  END; % of substituting parameters

  SKIPSEP;
  IF CURCH<>TTYEOL AND CURCH<>"^M" THEN
  BEGIN
    WRITE "Too many parameters";
    STOP;
  END;

  % Move tail along if necessary

  IF LBP<LENGTH[0] THEN % Need to move tail along
  BEGIN
    SPTR:=LBP; % Set search pointer to beginning of tail
    WHILE BUF[SPTR]<>"^M" DO
      SPTR:=SPTR+$01; % Find end of tail
    MPTR:=SPTR+LENGTH[0]-LBP;
    IF MPTR>=$80 THEN
    BEGIN
      WRITE "Insufficient room on command line";
      STOP;
    END;
    DO BEGIN
      BUF[MPTR]:=BUF[SPTR];
      MPTR:=MPTR-$01;
      SPTR:=SPTR-$01;
    END UNTIL SPTR<LBP;
    LBP:=MPTR+$01;
  END;
  % Copy string into line buffer

  LBP:=LBP-LENGTH[0]; % Backup LBP by string length
  MPTR:=LBP;
  SPTR:=$00;
  IF ECHOFL[0] THEN WRITE "+++";
  WHILE SPTR<LENGTH[0] DO
  BEGIN
    BUF[MPTR]:=STRING[SPTR];
    IF ECHOFL[0] THEN WRITE STRING[SPTR];
    MPTR:=MPTR+$01;
    SPTR:=SPTR+$01;
  END;
  BUFPTR:=BUFLOC+COMBINE($00,LBP)-$0001;
  LSTTRM:=TTYEOL;  % Tell flex more commands to go
END.
```

Editor's Note: For all those Whimsical users: here are some for you. And I might add that they are pretty handy utilities. I am still AMAZED at the quality of development software we are fortunate to have access to as compared to some of the other 'big'(?) systems. 68 never had it so good. If more folks knew what we have, we would grow faster than Jack's bean stalk!

And as the Mac community is concerned, they got a looooooong way to go!

NOTE: SIZE, PARSE, and CMD are available on 68 Micro Journal Reader Service Disk #18 (see p. 60).

DMW

A Follow-up to the Article in the July ´85 Issue

C.Dragon says...
Speed Things Up!

A Hex Dump In Quarter Time

By
Brad Taylor
Tom Gilchrist

There are times when you want to speed things up. If you are doing a lot of writing to the screen, old printf() will slow ya down. Sure its great for formating, but what if you just want to print out a word here, a string there? The hdump.c program will set you free...

Run Time	Program
10 secs	DUMP in 6809 asmb.
15 secs	hdump.c
46 secs	hex.c (a sample program from INTROL)

The times are for using the hdump.c source on a 1mhz system. Times are not exact (I counted, one thousand one, one thousand two, etc). The program was compiled on INTROL C under FLEX. Compile and enjoy!

```
/*
***** FDUMP -- dump a file in hexadecimal that's FAST! <<<<<<<<<<<<<<<<<<<<<<<
*
*    FDUMP <FILENAME>
*
*    Version 1.3
*    Written by: Bradford Taylor
*                Sharp Engineering
*    Date:       March 17, 1985
*
*    Although there are several file dump tools available for flex,
*    I never liked the output format.  Several "C" versions of file
*    dump utilities have been published in various magazines, but
*    these versions all used "printf" which slowed the output.  This
*    version was written to satisfy both readability and speed requirements.
*
*    This program is meant to be compiled with the FLEX port of the
*    INTROL C compiler. To compile under other compilers, you may need
*    to change the open() error.
*
*    This code is available on c.dragon (316) 943-9716 on disk FLEX #2
*    in the directory "sharp".
*
*/
#include <stdio.h>
#define NOCCARGC 1
#define MODE "rb"                        /* "rb" for FLEX, "r" others */
main(argc,argv)
int argc,*argv;
{

        int input,c,i,spaces;
        char *address,*pntr,asc[17];

        if(argc != 2)
         {
                outstr("usage: fdump <filename>\n");
                exit();
         }
         if((input = fopen(*++argv,MODE)) == ERROR)
          {
                 outstr("Can't open ");
                 outstr(*argv);
                 outstr(".\n");
                 exit();
         }

        *(pntr = asc) = address = 0;

        do
         {
          if(((int)address&255)==0)
           header();
          showword(address); putchar(':');
          spaces = 16*3+5;
          for(i=0;i<16;++i)
          {
           if((i&3)==0)
            {
              putchar(' ');
              --spaces;
            }
           if((c=getc(input)) == EOF)
             break;
           showbyte(c); putchar(' ');
           spaces -= 3;
           ++address;
           if(c<32 || c>127) c = '.';
           *pntr++ = c; *pntr = 0;
          }
          while(spaces--) putchar(' ');
          outstr(asc); crlf();
          *(pntr = asc) = 0;
         }
        while(c != EOF);

        fclose(input);

}


crlf()
{
        putchar('\n');
}

outstr(s)
char *s;
{
        while(*s)
         putchar(*s++);
}

header()
{

 crlf();
 outstr("ADDS  00 01 02 03  04 05 06 07  08 09 0A 0B  0C 0D 0E 0F     (ASCII)");
 crlf();
 outstr("----  -- -- -- --  -- -- -- --  -- -- -- --  -- -- -- --     -------");
 crlf();

}
showword(v)
        int v;
        {
        showbyte(v>>8);
        showbyte(v);
        }

showbyte(v)
        char    v;
        {
        showlow(v>>4);
        showlow(v);
        }

showlow(v)
        char    v;
        {
        v &= 0x0F;
        v += '0';
        if(v > '9')
                v += 7;
        putchar(v);
        }
```

IVA Electronics

6117 GERARD MORISETT, MONTREAL, QUEBEC H1M 3J8 – TEL (514) 256-0427

Dear Don

Thanks for a wonderful journal. You've done a great job of covering the sound engineering and innovative ideas found in the 6809 arena. Personally speaking, I have a 'E' board Coco which underwent many modifications since I encountered your publication. I think it was the 6809 and 68 Micro Journal which taught me enough to be a newly graduated honours electrical engineer (computer option). Couldn't have done it without you guys out there!

My memories of back then (when you could only buy a 16K Coco and be happy with it) were fond. My first basic program was to estimate the missing variable in one of those JFET source-drain equations. This soon progressed into Lunar landing games and 3-D equation surface plots. (boy those high res pixels take long to pop up undergoing the rotation matrix I was hitting them with) In any case, along came flex with Lucidata pascal as the icing. What a combination! Thank god for the paged mode run time P code interpretation. I managed to run some programming assignments from my prog courses at home. The markers didn't mind as long as ISO standards were adhered to.

It was then that I experienced a peculiar disk problem. I would run my disks all night long but given a day later, some missing links will inevitability show up. This had me changing drives and frequent alignments. The consistant culprit was due to the fact that I had placed my drives inbetween the CRT monitor and the phone. Can you imaging the flux density going thru my 5 1/4 round babies! Talk about inadvertent erasure! The solution was to spread eagle, all my computer components on my work bench. If someone out there is experiencing it, well what can I say.

What's my system now? 64K OS-9 based Coco mainly running C programs. Its PIC so why not? My interest now is to develop a batch supervisor program which executes 'batch jobs' using other 64K banks. (everyone should have heard about the 128K boards by now) You should be able to trap all reasonable service calls and pass them with a little or no modifications to the kernel. The problem lies in processes which want information or modifications done to their memory environment. Thats a sticker yet to be figured. Any ideas out there? Given the 128K cards (and Maxsys offering more!), I don't see why someone hasn't come up with a OS-9 level two kernel to take advantage of the new capability. RAM disks are a cop out effort to the new challenge. If someone provides me with the capability of running large programs in multiuser mode, I'll invest in memory expansions with a hard disk drive for sure. Right now, running this watered down version of OS-9 on the Coco is perplexing. (I even had to pay $50 for the standard OS-9 ACIA and PIA source code on a disk).

One thing that has bothered me. I don't know how an actual disk driver works. I know all about the WD family of FDCs but never took the time to investigate further. Will some kind soul with a copy of a workable driver please send one to me. Waves of gratitude will befall you! Also, as my predominant area of interest in computers are Fault tolerant computing, parallel architectures and VLSI automated design, I welcome friendly cooresspondence with parties having similiar interests. I have just finished a research project into robotic collision avoidance. Once again, thanks a million.

Jack Tay

Dear Sirs,

First let me tell you that we enjoy every single issue of your '68' micro journal. We are even trying to start a regular contribution from a very enthousiastic UniFLEX user group here in Holland. We think that UniFLEX is really tremendous but too little pages are used for it in '68'. Maybe this is because TSC keeps all nice things hidden from the users (how to write and implement own drivers etc.). TSC must be aware of the fact that Microware (OS-9) is much more open about this. It's about time that they change their policy. Hopefully you agree with us and will pass the word...
Secondly I want to mention that we have KERMIT (file transfer utility) available for FLEX and UniFLEX. The first is an adapted version of the UNIX C-version which can be compiled with INTROL-C. The UniFLEX version IS the UNIX C version (you see how portable C is) and can be compiled using the McCosh compiler. Both are around 1200 lines source. If you are interested I will try to write an abstract from the protocol manual for publication.
Speaking of data communication, is it possible to publish a list of BBS phone numbers with regular updates somewhere in '68'?
The CS/SWTPc usergroup can be reached at address at the bottom of this letter.

Best regards,

CS/SWTPc User Group,
Egbert Jan van den Bussche,
Room 5B-a,
2611 LV DELFT
HOLLAND phone (0)15-144552

Editor's Note: Egbert, Thanks for the offer of software material. I know that thousands of readers, worldwide, would appreciate very much your contributions. We would run more UniFLEX material but I receive little if any cooperation from TSC, and little UniFLEX input otherwise. I guess they are satisfied with things as they are, and most users are too busy, or something. Or it just might be that there are a lot more OS-9 users. However, you are right, OS-9 is better supported, so my readers say. And we certainly get far more OS-9 material than UniFLEX. Fact is I have run every bit of UniFLEX material received todate.

I think one difference is that TSC never did (despite promising over 5 years ago) to come out with the 'configurable version'. I was repeatly told that the code was finished and they were just waiting to get the documentation done. It has now been five years and still not released. Must be one big mother of a manual!

I will attempt to get an updated BBS listing in a future issuer of 68 MICRO JOURNAL. I would appreciate any input on this from others who know of any BBS with its number, that our readers would be interested in.

DMW

GIMIX INC. 1337 WEST 37th PLACE • CHICAGO, ILLINOIS 60609
(312) 927-5510 • TWX 910-221-4055

Introducing the GMX-020™ - MC68020 Processor Board from GIMIX

MC68020 32-Bit Microprocessor

The GMX-020 CPU Board uses the state-of-the-art MC68020, the newest and most powerful member of Motorola's M68000 family of microprocessors. The MC68020 is a full 32-bit processor with seperate 32-bit address and data buses, an on-chip instruction cache, and a coprocessor interface. The MC68020 is object-code compatible with earlier members of the M68000 family, with enhancements to the instruction set providing additional support for high-level languages and systems software. The processor also supports demand-paged virtual memory.

The pipelined internal architecture of the MC68020 allows overlapping execution of instructions, and can result in a net instruction execution time of zero under certain circumstances. This, along with the on-chip cache and other enhancements make the processor typically 400% more powerful than its predecessors. The 16 MHz version can process instructions at a sustained rate of 2 to 3 million instructions per second (MIPS) and at burst rates exceeding 8 MIPS.

GMX-020 Processor Board

The GMX-020 CPU Board is designed for maximum utilization of the power of the MC68020, while retaining compatibility with the already proven GIMIX line of peripherals such as DMA disk controllers and intelligent I/O processors. The board features:

- An MC68020 processor operating at a 12.5 MHz clock rate (16.5 MHz optional when available)
- A 4K byte (1K long word) instruction-only physical address cache operating at full processor speed (no wait-states). The on-board cache can be operated in any one of four modes to optimize cache utilization for a particular operating system or application. The cache RAM can also be used as high-speed (no wait-state) RAM when the cache is not enabled.
- A high-speed, discrete Memory Management Unit (MMU) that supports multi-user, multi-tasking operation and demand-paged virtual memory environments. Use of the MMU causes no increase in memory access time. In addition to dynamic address translation, the MMU associates four seperate attributes with each 4K segment of memory: a write-enable bit to protect shared text; a "valid" bit to flag segments containing valid data, a "dirty" bit to flag segments that have been modified, and an "access" bit to indicate that a segment has been used. The standard MMU configuration supports 4 Megabytes of virtual memory with up to 16 seperate segment maps. Other configurations can allow up to 8 Mbytes of virtual memory, or up to 64 seperate maps.
- An optional floating-point coprocessor (MC68881) that directly extends the architecture and the instruction set of the processor to include floating-point data types, full support for IEEE Rev 10.0 high level math functions, and also transcendental and other standard math functions. All coprocessor calculations are performed to 80 bits of precision.
- Six levels of prioritized autovector interrupts from seven sources. Two interrupt sources are internal to the board, three are from the bus, and two (non-maskable) are from sources connected directly to the CPU board.
- Three seperate hardware prioritized channels for external DMA devices. Simultaneous DMA requests on different channels are arbitrated by the board on a channel priority basis.
- A 20-bit external address bus for up to 1 Megabyte of physical memory space (RAM and I/O). The I/O devices occupy the upper 4K bytes of the 1 Mbyte address space. Two seperate areas are defined within the I/O space, each with optimum timing for particular I/O devices. (Note: The I/O timing will not support any 6800/6809 peripheral devices such as the 6850 or 6821. Serial and parallel I/O is supported only through GIMIX intelligent I/O processors.)
- Two EPROM sockets that accept 8K, 16K, 32K, or 64K x 8-bit industry standard devices for up to 128K bytes of on-board firmware. The EPROMs are addressed above the 1 Mbyte RAM space, with auto-mapping of the restart vectors to low memory on power-up or reset.
- A full-featured hardware time-of-day clock/calendar with battery backup, which can also generate interrupts at one second intervals.
- A seperate "tick" generator that can generate interrupts at precise, jumper selectable intervals ranging from 10 microseconds to 20 minutes. Interrupts from the "tick" generator can be enabled or disabled under program control, and have their own priority level to minimize overhead during context switching.
- A seperate voltage regulator board that powers the board and provides standby battery power for the TOD clock. The regulator board receives its input from the standard power supply in the GIMIX mainframe.

St. James's University Hospital
Beckett Street. Leeds LS9 7TF
Telephone 0532-433144

NON-KEYBOARD DATA ENTRY TO A 6800 MICROCOMPUTER

A J Hall
St James's University Hospital
Beckett Street   Leeds LS9 7TF

INTRODUCTION

Digitisers or graph tablets convert graphical coordinate data from a diagram, map, chart recording or menu into digital form for analysis or processing by computer. Coordinate points, which must lie within the active area of the digitiser, are specified by touching them with a pen-like stylus. The Summagraphics "Bitpad One" (Trade Mark) is an example of what is available commercially; it operates on the magnetostrictive principle with a current pulse being sent along a "send" wire lying at right angles to a magnetostrictive wire mesh laid beneath the pad surface. The current causes the mesh dimensions to change and the resultant strain wave is detected by coils within the stylus. An inbuilt microprocessor calculates the coordinate position of the pen from the time taken for the wave to reach the stylus, it also senses whether the stylus tip microswitch is closed or not and sets a flag accordingly. Depending on the type of Bitpad the data is then output, either in 8 bit parallel or in RS232 serial format.

MODES OF OPERATION

The mode and rate of sampling data points can be predetermined by internal switch settings or be selected under software control; the latter is more flexible especially if the bitpad is interfaced to a microcomputer. The RS232 interfaced Bitpad is controlled by sending an ASCII character to it: this selects the desired sampling rate of coordinate pairs and the operation mode. The modes are as follows:-

Point Mode - on depressing the stylus and closing the tip microswitch the bitpad outputs an X, Y coordinate pair and flag for stylus up/down.

Stream mode - the flag and coordinate pair are output continuously while the stylus is either in contact with the pad or close to its surface.

Switch Stream Mode - depression of the stylus to close the tip microswitch causes a stream of coordinate pairs and flags to be output until the stylus is lifted to open the tip microswitch even though the tip still remains in contact with the pad.

Data Format - RS232 serial and arranged as below

XXXX, YYYY,F CR LF

The line feed (LF) is optional and switch selectable: the data is in ASCII BCD format where X = X axis value Y = Y axis value, F = 0 or 1 and flags the state of the tip microswitch.

INTERFACING THE BITPAD

I have used an RS232 bitpad connected to a SWTP 6800 microcomputer via a serial interface at Port 0. The baud rate and stop bits of the Bitpad should, of course, correspond to that of the interface. Commas are used to seperate each block of coordinate and flag data and the string is terminated with a carriage return. The format is identical to the keyboard input expected in response to a Basic "Input" statement so to communicate with the bitpad only the I/O vector in SWTBUG has to be changed - from Port 1 to Port 0. This is done by a "POKE" statement to alter the contents of $A00B (40971 decimal) from 04 to 00. This alters the port address from $8004 to $8000.

Directly accessing the Bitpad from your Basic program in this manner means that only the point mode can be used: it generates one set of coordinate data when the stylus is depressed, the other modes produced strings of data and these will not be acceptable. So machine code subroutines are necessary utilising SWTBUG INEEE input routine and detecting the carriage return to seperate the coordinate pairs and flags. However for many purposes the point mode is all that is needed eg selecting options on a menu or specifying specific points on a curve.

At the end of any access to Port 0 the I/O vector should be reset to port 1 so that the keyboard can be used for debugging the program during development. The routines are as follows:

SELECT POINT MODE

This is done once at the start of your program unless you intend to change modes while the program is running, in which case it should be called a subroutine.

```
POKE (40971,0)  REM SELECT PORT 0 I/O
PRINT "P"       REM P sets pad to point mode
POKE (40971,4)  REM I/O to Port 1
```

As we are working in Basic the port addresses have to be in decimal unless your Basic allows hex characters.

INPUT OF DATA FROM BITPAD

Having selected Point mode a general routine is needed to get data from the pad. One is given below.

```
POKE (40971,0)  REM I/O to Port 0.
INPUT X,Y,F     REM wait for input
POKE (40971,4)  REM I/O back to Port 1.
```

Thus a few simple Basic commands allow the Bitpad to be accessed and data obtained for a program to operate on.

DATA ENTRY USING THE BITPAD

One example is the analysis of multiple choice questionnaires where the answers are indicated by ticks in the appropriate boxes. The form is placed against reference marks on the Bitpad so that the locations of the answer boxes are defined. One approach, which ensures that your program will always respond in a predictable manner, is to divide the area or areas within which the answer boxes lie into the elements of a basic two dimensional array which I will call P(Y,X)(see fig 1. On entry to the program the array is filled with 1's and then specific locations are set with numbers in ascending order (2,3----etc). Each number corresponds to a specific answer and we can control program branching using the Basic instruction on X GOTO---. If a blank area is accidentally touched P(Y,X) gives a 1 and the first address in the On X GOTO instructions is a routine which gives an error message and awaits further input. The Bitpad has an active area of 11" x 11" and a resolution of .0005", thus the coordinates can range from 0-2200 in both X and Y with the origin at the bottom lefthand corner. The values returned by the Bitpad will lie in this range and they must be scaled so that integer values of X and Y are produced to address the array P(Y,X). A typical example is given below.

```
    X = X - XO              REM XO, YO are the answer area origins.
    Y = Y - YO              REM X, Y are coordinates from stylus.
    X = X/S                 REM S is the length of one side of the
    Y = Y/S                 REM of the answer square in .0005 units.

    X = INT(X)+1            scale answer from 1 up
    Y = INT(Y)+1
    X = P(Y,X)
    ON X GOTO----
```

Additional traps can be inserted before the statement X=P(Y,X) to handle values greater or less than the array dimensions.

SUMMARY

The interfacing of a serial version of a Summagraphics Bitpad via a MP-S interface located at Port 0 is simple and straightforward and it can be accessed directly from a Basic program by simple statements. The Bitpad will have to be set to suit the configuration of Port 0- some Basics set it to a configuration that is different to the SWTBUG configuration. If the Bitpad is used for data entry then a Basic array is a useful means of ensuring that your program behaves in a predictable manner if the stylus is placed in an area where there are no answer boxes. In addition a program can easily be altered to accommodate additions or deletions. Deletions require the appropriate value in the array to be replaced by a 1 while additions require the replacement of a 1 by the next free number in the ON X GOTO sequence and the addition of code to deal with the response. The examples given for generality have altered the I/O vector by means of POKE statements to access Port 0 but many Basics provide a "Port" statement to do this. However if you are using SWTP's Basic version 3.5 the following corrections are needed before the "PORT" statement can be used. Location $149E should be altered from $34 to $30 and $1721 from $C2 to $C8.

MICRONICS
RESEARCH CORP.

33383 LYNN AVENUE,
ABBOTSFORD,
BRITISH COLUMBIA,
CANADA, V2S 1E2

Dear Don,

This month's offering represents quite a departure from those submitted in the past, in that matchiog changes have had to be made to the trio FLEX, COPY and CAT.

I began by writing an overlay for FLEX to allow the TIME of a file's creation to be stored in the Directory, as well as the date. This obviously meant that COPY also had to be modified to preserve the time-code during copying, plus CAT in order to display the decoded time-byte in the form 17:24. Encoding of the time-byte is compatible with the "standard" developed by Peter Stark in his TCAT (which, by the way, will still be needed to time-SORT a catalog), with the exception that my modification can distinguish between midnight (0:00hrs) and a non-encoded file.

As I use only Leo Taylor's CAT and COPY utilities, I've modified just these two to match with the FLEX overlay. The patch to COPY is minimal but unfortunately, in order to eccommodate all the extra code necessary to patch CAT and still squeeze it into the Utility-Command area, I've had to completely re-write it in 6809 Assembler. So 6800 users are out of luck, as I don't think it's advisable to relocate CAT in low memory. I ended up with enough room to incorporate a Random-File indicator as well. These goodies will, of course, only be displayed if the 'D' or 'M' options of CAT are exercised. The end result is that CAT is really too big to merit re-printing again in 68MJ, (though this is entirely up to you) so I'm sending it to you on disk, together with the FLEX overlay and the completely patched COPY command. The new CAT also displays the Disk-Name AND Extension, while a matching upgraded DATE command permits changing this extension. Sorry about that, Leo, and apologies to Ron Anderson too!!

Readers may therefore obtain the complete set from your disk library, though in order to get them going it might be as well to publish CLOCK.ASM, which should be assembled as CLOCK.OVL, and the patch to COPY, which is :

```
Locate NOTRND in Leo's COPY utility and patch
NOTRND TST UPDFLG
       BNE WRTNOR
       LDAA SRCFCB+24 Get Time-Code    * New Code to
       STAA 24,X and save for copying  * be inserted
       LDAA SRCFCB+25 ....... etc
```

This way they can begin time-coding their files ready for when they obtain the upgraded CAT to display it.

Sincerely,

Bob

R. Jones
President

PS No free subscriptions, please! It's MUCH more important to keep 68MJ going. Like your other reader I wish it were a weekly journal!!!!

To fill up the disk I've thrown in a few other utilities which I've enhanced in some way, PDEL in collaboration with Leo Taylor.

Editor's Note: *Without a doubt, I am the most fortunate guy in the world. I can't think of another journal that has such unselfish suport as we do.*

This set will be offered as "68 Micro Journal Readers Service Disk number 19." I wonder how many of you readers realize what a real bargain these service disk are? This disk alone has utilities that if they were purchased just as binary utilities, they would sell, say, on the PC market for over $300-400 bucks! Thats right, their stuff is that expensive, and in most cases, they don't even have this sort of stuff. And here you even get source!

On the Apple Mac(c) market they just don't have anything like this.

Bob, all I can say is - THANKS, FROM ALL OF US HERE AT 68 MICRO JOURNAL AND THOUSANDS OF OTHER READERS, WORLDWIDE!!!!!!!!!!

Below is a directory of disk number 19.

DMW

```
DRIVE:2  VOLUME:MICRO68  1  CREATED:14-JUN-85

FILE#  NAME     TYPE BEGIN  END   SIZE   DATE    PRT

    1  CLOCK    .ASM 01-01  01-05    5  13-JUN-85
    2  DATE     .ASM 01-06  04-08   33   1-NOV-84
    3  COPY     .ASM 04-09  12-02  134   5-JUN-85
    4  CAT      .ASM 12-03  17-09   57  11-JUN-85
    5  PDEL     .ASM 17-0A  1A-01   22   8-DEC-84
    6  PDEL     .DOC 1A-02  1A-0A    9   8-DEC-84
    7  ERRORS   .SYS 1B-01  1D-0A   30  11-SEP-82
    8  DO       .ASM 1E-01  22-05   45   7-MAY-85
    9  DO       .DOC 22-06  24-02   17   7-MAY-85
   10  LOG      .ASM 24-03  25-08   16  31-MAY-85
   11  LOG      .DOC 25-09  26-04    6  11-NOV-84

FILES=11  BIGGY=134  TOTAL=374/374  FREE=16
```

UNIBOARD Computer Notes

Eric Sillanpaa
134 Pleasant Drive
Sault Ste. Marie, Ontario
Canada P6B 4E2

Dear Don,

I have a UNIBOARD computer that I purchased from Digital Research Computers and I am very happy with it. I purchased the bare board and populated it myself.

Here are some hints that other owners may find helpful. A bell can be added by tying a .5 second one shot to U26-7. The one shot should drive an invertor that will be used as a buffer to drive a small solid state alarm such as a Sonalert. Control G will then sound the bell.

The Uniboard manual implies that the board will operate with either 5 or 8 inch drives just by changing the setting on the configuration dip switch. This is not correct. The board will not work correctly with 5 inch disks as it is. R33 and C41 must be changed from 33 ohms and 0.33 mfd. to 68 ohms and 0.68 mfd. It is also important that RP6 be 150 ohms. This should cure intermittent double density read problems.

The Unimon monitor is not provided with a routine to select sides even though the side select line is decoded. I added the side select routine to Seek as shown in the listing. I also modified Newdisk the way S. D. Lyon did in June '85 M.J. to work with 18 sectors per side. I did not use any routines past write track because they are on the Rom.

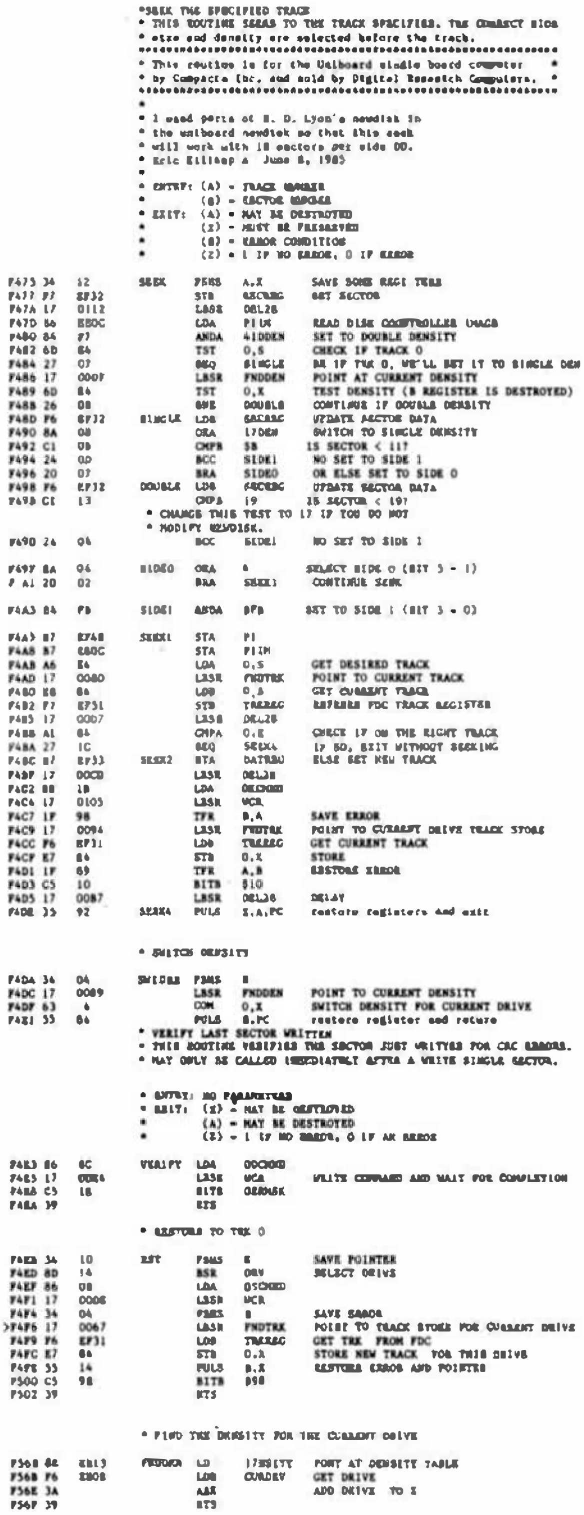
I had no 2732 Eproms so I used a 2764 by programming the last half and then strapping pins 1,2,38,39,40 together. 1,2,39, and 40 are then either cut off or bent out and it is then placed in the socket at U14.

I use two Shugart SA455 DS/DD half height disk drives and I am very happy with them. It is nice to have 1404 sectors available per drive.

Would you or any of your readers know what determines the stepping rate for the disk heads? My drives will operate at 6ms. but I have not found the correct time delay to reduce yet.

Yours truly,

Eric



```
*SEEK THE SPECIFIED TRACK
* THIS ROUTINE SEEKS TO THE TRACK SPECIFIED. THE CORRECT SIDE
* size and density are selected before the track.
****************************************************************
* This routine is for the Uniboard single board computer      *
* by Compacta Inc. and sold by Digital Research Computers.     *
****************************************************************
*
* I used parts of S. D. Lyon's newdisk in
* the uniboard newdisk so that this seek
* will work with 18 sectors per side DD.
* Eric Sillanpaa  June 8, 1985
*
* ENTRY: (A) = TRACK NUMBER
*        (B) = SECTOR NUMBER
* EXIT:  (A) = MAY BE DESTROYED
*        (X) = MUST BE PRESERVED
*        (B) = ERROR CONDITION
*        (Z) = 1 IF NO ERROR, 0 IF ERROR

F475 34    12            SEEK    PSHS   A,X        SAVE SOME REGISTERS
F477 F7    EF32                  STB    SECREG     SET SECTOR
F47A 17    0112                  LBSR   DEL28
F47D B6    EE0C                  LDA    PIIM       READ DISK CONTROLLER IMAGE
F480 84    F7                    ANDA   #IDDEN     SET TO DOUBLE DENSITY
F482 6D    E4                    TST    0,S        CHECK IF TRACK 0
F484 27    07                    BEQ    SINGLE     BR IF TRK 0, WE'LL SET IT TO SINGLE DEN
F486 17    00DF                  LBSR   FNDDEN     POINT AT CURRENT DENSITY
F489 6D    84                    TST    0,X        TEST DENSITY (B REGISTER IS DESTROYED)
F48B 26    08                    BNE    DOUBLE     CONTINUE IF DOUBLE DENSITY
F48D F6    EF32          SINGLE  LDB    SECREG     UPDATE SECTOR DATA
F490 8A    08                    ORA    #SDEN      SWITCH TO SINGLE DENSITY
F492 C1    0B                    CMPB   $B        IS SECTOR < 11?
F494 24    0D                    BCC    SIDE1      NO SET TO SIDE 1
F496 20    07                    BRA    SIDE0      OR ELSE SET TO SIDE 0
F498 F6    EF32          DOUBLE  LDB    SECREG     UPDATE SECTOR DATA
F49B C1    13                    CMPB   19        IS SECTOR < 19?
                * CHANGE THIS TEST TO 17 IF YOU DO NOT
                * MODIFY NEWDISK.
F49D 24    04                    BCC    SIDE1      NO SET TO SIDE 1

F49F 8A    04            SIDE0   ORA    4         SELECT SIDE 0 (BIT 3 = 1)
F A1 20    02                    BRA    SEEK1      CONTINUE SEEK

F4A3 84    FB            SIDE1   ANDA   $FB       SET TO SIDE 1 (BIT 3 = 0)

F4A5 B7    EF48          SEEK1   STA    PI
F4A8 B7    EE0C                  STA    PIIM
F4AB A6    E4                    LDA    0,S        GET DESIRED TRACK
F4AD 17    00B0                  LBSR   FNDTRK     POINT TO CURRENT TRACK
F4B0 E6    84                    LDB    0,X        GET CURRENT TRACK
F4B2 F7    EF31                  STB    TRKREG     REFRESH FDC TRACK REGISTER
F4B5 17    00D7                  LBSR   DEL28
F4B8 A1    84                    CMPA   0,X        CHECK IF ON THE RIGHT TRACK
F4BA 27    1C                    BEQ    SEEK4      IF SO, EXIT WITHOUT SEEKING
F4BC B7    EF33          SEEK2   STA    DATREG     ELSE SET NEW TRACK
F4BF 17    00CD                  LBSR   DEL28
F4C2 86    1B                    LDA    SKCMD
F4C4 17    0105                  LBSR   WCR
F4C7 1F    98                    TFR    B,A        SAVE ERROR
F4C9 17    0094                  LBSR   FNDTRK     POINT TO CURRENT DRIVE TRACK STORE
F4CC F6    EF31                  LDB    TRKREG     GET CURRENT TRACK
F4CF E7    84                    STB    0,X        STORE
F4D1 1F    89                    TFR    A,B        RESTORE ERROR
F4D3 C5    10                    BITB   $10
F4D5 17    00B7                  LBSR   DEL28      DELAY
F4D8 35    92            SEEK4   PULS   X,A,PC     restore registers and exit


                * SWITCH DENSITY

F4DA 34    04            SWIDEN  PSHS   B
F4DC 17    0089                  LBSR   FNDDEN     POINT TO CURRENT DENSITY
F4DF 63    84                    COM    0,X        SWITCH DENSITY FOR CURRENT DRIVE
F4E1 35    84                    PULS   B,PC       restore register and return
                * VERIFY LAST SECTOR WRITTEN
                * THIS ROUTINE VERIFIES THE SECTOR JUST WRITTEN FOR CRC ERRORS.
                * MAY ONLY BE CALLED IMMEDIATELY AFTER A WRITE SINGLE SECTOR.

                * ENTRY: NO PARAMETERS
                * EXIT:  (X) = MAY BE DESTROYED
                *        (A) = MAY BE DESTROYED
                *        (Z) = 1 IF NO ERROR, 0 IF AN ERROR

F4E3 86    8C            VERIFY  LDA    RDCMD
F4E5 17    00E6                  LBSR   WCR        WRITE COMMAND AND WAIT FOR COMPLETION
F4E8 C5    18                    BITB   ERMASK
F4EA 39                          RTS

                * RESTORE TO TRK 0

F4EB 34    10            RST     PSHS   X          SAVE POINTER
F4ED 8D    14                    BSR    DRV        SELECT DRIVE
F4EF 86    08                    LDA    RSCMD
F4F1 17    00D6                  LBSR   WCR
F4F4 34    04                    PSHS   B          SAVE ERROR
>F4F6 17   0067                  LBSR   FNDTRK     POINT TO TRACK STORE FOR CURRENT DRIVE
F4F9 F6    EF31                  LDB    TRKREG     GET TRK FROM FDC
F4FC E7    84                    STB    0,X        STORE NEW TRACK FOR THIS DRIVE
F4FE 35    14                    PULS   B,X        RESTORE ERROR AND POINTER
F500 C5    98                    BITB   $98
F502 39                          RTS


                * FIND THE DENSITY FOR THE CURRENT DRIVE

F568 8E    EB13          FNDDEN  LD     DENSITY   POINT AT DENSITY TABLE
F56B F6    EB08                  LDB    CURDRV     GET DRIVE
F56E 3A                          ABX               ADD DRIVE TO X
F56F 39                          RTS
```

ERRORS.SYS Patch

Dear Don,

Please find enclosed a patch to FLEX that cures a rather annoying flaw in this otherwise great piece of software. Every time some program calls a file that can't be found, this will be reported, but there is no way to tell the name of the file, that the program is looking for. This is for example the case when the assembler is looking for a LIB-file which isn't there.

The reason is that RPTERR uses the system FCB when it translates the error type to a message in ERRORS.SYS, and since the file not found also was in this FCB, all information is lost.

The cure is rather simple: Use another FCB in the RPTERR routine. Below is a small patch (actually only two bytes need to be changed) to implement it. I use the FCB starting at $CAC0 that otherwise is used by the spooler, but any free area may be used. There may also be changes from one version of FLEX to another, but I assume you could always do like this:

First find the address of the RPTERR routine in $CD3F, and then follow the listing below until you find the correct place to make the change. This has to be done directly on disk using a program like DISKEDIT or REPAIR.

Following is the routine as it is implemented in my version of FLEX:

```
                        .
                        .

39          RNDERR  RTS

            * Report error routine

A6   01     RERR    LDA   1,X       Check error status
B7   CC20           STA   ERRTYP    and save it
27   F8             BEQ   RNDERR    Return if no error
34   30             PSHS  X,Y
BD   CDE7           JSR   RESIO     Standard I/O
108E CC2D           LDY   ERRVEC    Use 'ERRORS.SYS' ?
26   08             BNE   NOTSYS    No, special file
81   10             CMPA  #16       Drives not ready?
27   53             BEQ   NTRDY     Report it
108E D39C           LDY   #ERRFIL   Point Y to 'ERRORS.SYS'
8E   C840   NOTSYS  LDX   #SYSFCB   System FCB
A6   02             LDA   2,X       File open?
27   09             BEQ   NOOPEN
86   04             LDA   #4        If so, close it
A7   84             STA   ,X
BD   D406           JSR   FMS
26   28             BNE   RDSKER    Disk error
8E   CAC0   NOOPEN  LDX   #ERRFCB+4 Use another FCB
C6   0B             LDB   #11       Put in file name
8D   66             BSR   COPYTX
                        .
                        .
45 52 52 4F ERRFIL  FCC   "ERRORS",0,0,"SYS"
                        .
```

To have a report of the name in the system FCB, the following routine may be used. It naturally has to be resident and is linked to the list of system commands as described in the FLEX users manual.

```
8E   D3A7   OCMD    LDX   #OMSG       Show message
BD   CD1E           JSR   PSTRNG
8E   C843           LDX   #SYSFCB+35
A6   80             LDA   ,X+         Get drive #
8B   30             ADDA  #'0         Make ASCII
BD   CD18           JSR   PUTCHR      Show it
86   2E             LDA   #'.
BD   CD18           JSR   PUTCHR
C6   08             LDB   #8          Show name
8D   0C             BSR   ONAMLP
86   2E             LDA   #'.
BD   CD18           JSR   PUTCHR
C6   03             LDB   #3          Show extension
8D   03             BSR   ONAMLP
7E   CD03           JMP   WARMS
A6   80     ONAMLP  LDA   ,X+         Get character
BD   CD18           JSR   PUTCHR      Show it
5A                  DECB              Decrement counter
26   F8             BNE   ONAMLP
39                  RTS

46 69 6C 65 OMSG    FCC   "File in FCB: ",EOT
```

I hope this may be of use to others who are usin FLEX.

Yours sincerely.

Niels Oesten
Broendbyvestervej 50 1. tv.
DK-2600 Glostrup
Denmark.

MORNING ADVOCATE
STATE-TIMES
SUNDAY ADVOCATE

BOX 588 BATON ROUGE, LA 70821-0588 PH. (504) 383-1111

Gentlemen:

Anyone interested in a 6805 one chip computer user group, please contact me at the address below.

We are currently in the process of writing a real time multitasking kernel.

Sincerely,

Bruce Vicknair
State-Times & Advocate
P. O. Box 588
Baton Rouge, LA 70821
Telephone: 504/383-1111, ext. 163

SARDIS
TECHNOLOGIES
2261 E. 11th Ave.
Vancouver, B.C., Canada V5N 1Z7

NEW PRODUCT ANNOUNCEMENT

512K RAM Expansion
for the ST-2900

Sardis Technologies are pleased to announce a major addition to their 6809 based ST-2900 system. The new RAM-512 board plugs into the ST-2900's expansion connector to add a half-megabyte of memory. The new memory board fits between the existing CPU and FDC boards to form a very compact three board "sandwich".

Greatly improved system performance results from using the added memory as a high-speed virtual disk. The huge storage capacity — 40% more than a double-sided, double-density 40 track 5" disk — handles large sorts and compiles with room to spare. RAM-DISK driver routines and memory diagnostic utilities for FLEX or OS-9 are included with each board. Also currently under development for the RAM-512 are new disk driver routines for FLEX to dramatically speed up reading and writing to floppy disk.

The RAM-512 uses a proprietary dynamic RAM refresh circuit, based on the odd/even technique, that guarantees all memory locations are refreshed well within specifications, but avoids the severe "over refreshing" found on some other designs that unnecessarily increases power consumption.

First deliveries are scheduled for September 1985, with the price expected to be under ($US) $495. Please contact Dave Wiens at the address above for more details, or phone (604) 255-4485 regarding delivery and pricing.

Incompatible Disks - Another Solution

The article entitled, "Incompatable Disks", by S.D. Lyon in the June 1985 issue, page 38 described the problems using Peripheral Technology's PT-69 computer to read disks which were not formatted on that computer.

I also have run into problems using my PT-69 Computer to read double sided or double density disks. My problem was compounded by the fact that my PT-69 computer has a Western Digital 2797 floppy disk controller. This chip has a side select bit in the command register which MUST match the side code found in the Identification Mark on the disk it is reading. As S.D. Lyon pointed out, the output from the side select bit is used by Peripheral Technology to select the proper density for the reading. The upshot of these requirements is that it is impossible to read normal, single density, double sided IBM format or OS9 format disks on my PT-69 computer. This is because the side select bit must be set to $01 to match the side code on the disk AND it must also me set to $00 to cause the 2797 to read the disk in single density.

To solve this problem I took a different tack from S.D. Lyon. Rather than rewriting the disk formatting program to create non-standard disks, I modified the hardware of my PT-69. The modification requires no additional integrated circuits and, in fact, uses one less gate than the original design. The purpose of the changes are to allow the side and density to be selected separately from one another.

The necessary changes are shown on the enclosed diagram. The solid lines show the original circuitry; the "x's" show cuts in the existing traces and the dashed lines are the added wires. All of the cuts and patches can be made on the bottom of the PT-69 board. I used wire wrap wire for the three new connections.

Once the changes are made, D6 in the Drive Register ($E014) will select side (0 = Side 0, 1 = Side 1) and D7 will select density (0 = Single Density, 1 = Double Density). This modification is compatable with the standard TSC boot as a reset clears the Drive Register and sets the controller to Side 0, Single Density.

The disk drivers used with the PT-69 will have to be modified so that they will properly set the Side and Density bits in the Drive Register. A set of disk drivers which I have used successfully with the modified PT-69 is below. This disk driver is based on DRIVERS.TXT by Leo Taylor. If you have the source code for the PT-69 patches to FLEX, you can assemble this code and incorporate it in a new FLEX.SYS file. If you do not have the PT-69 drivers source code, you can assemble these new drivers and then APPEND them on your current FLEX.SYS. When FLEX.SYS is loaded they will load over the top of the old drivers.

So far I have found no bugs in either the hardware or software modifications. I have used them to read both OS-9 and Radio Shack disks. The software automatically switches density as needed to read these disks. When I first tested the drivers, I tried to read a Radio Shack disk using my DR.CMD without knowing how it was formatted. I was able to read it without difficulty. It was only after I looked at the bytes in the driver which show the disk format that I discovered that the disk was double density.

If you waot a copy of the source code for the disk drivers, send me a blank disk, and a mailer with postage attached and I will send you a copy.

Ken Drexler
311 Wilson Way
Larkapur, Calif. 94939

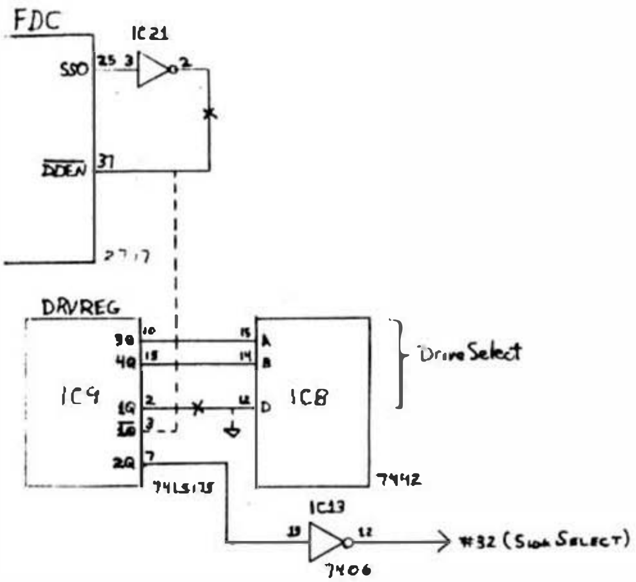
```
                          ****************************************
                          *
                          * DISK DRIVERS FOR FLEX 9
                          *
                          * File Name:  FDRVPT69.SRC
                          *
                          * Date: November 23, 1983
                          *
                          * Revised December 1, 1983 to use DRQ and
                          * INTRQ status bits
                          * Revised December 4, 1983 to add double
                          * density code.
                          * Revised December 27, 1983 to limit
                          * CHECK to the number of installed drives.
                          * Revised December 14, 1984 to correct code.
                          * Revised May 4, 1985, to conform to hardware
                          * changes on the PT69's side select and
                          * density logic.
                          *
                          * Designed for use with a PT-69 Computer
                          * using up to four 5 1/4 inch double
                          * sided, double density drives.
                          *
                          * DRIVE REGISTER USAGE:
                          *   D0, D1 - Drive select
                          *   D6     - Side select: 0 - Side 0
                          *   D7     - Density select: 0 = SD; 1 = DD
                          *
                          * Side Select is through D6 in the Drive
                          * Register: 0 = Side 0; 1 = Side 1.  Also
                          * the Command Register bit D1 MUST match the
                          * side byte in the disk's ID field.  On FLEX
                          * single density disks, this field is $00,
                          * regardless of the actual side.
                          *
                          * A read of the Drive Register returns
                          * DRQ as D7 and INTRQ as D6.
                          *
                          ****************************************

                          * FLEX9 EQUATES
                    CD00  COLDS   EQU    $CD00
                    CC34  PBFLAG  EQU    $CC34     1 = PRINT SPOOLER ACTIVE

                          * CONTROLLER EQUATES
                    E014  DRVREG  EQU    $E014     WD X79X CONTROLLER
                    E018  COMREG  EQU    DRVREG+4
                    E019  TRKREG  EQU    DRVREG+5
                    E01A  SECREG  EQU    DRVREG+6
                    E01B  DATREG  EQU    DRVREG+7

                          * MISCELLANEOUS EQUATES
                    0002  SSOBIT  EQU    %00000010 D1 IN COMMAND
                    0080  DNSBIT  EQU    %10000000 D7 IN DRIVE REG.
                    0040  SIDBIT  EQU    %01000000 D6 IN DRIVE REG.

                          * HIGHEST NUMBER DRIVE INSTALLED
                    0001  MAXDRV  EQU    1         DRIVE 0, 1

                          * FLEX DISK DRIVERS
                          *
 DE00                             ORG    $DE00     (AVAIL. THRU $DFBF)
>DE00 7E      DE2E        READ    JMP    READ1     READ SECTOR
 DE03 7E      DEBE        WRITE   JMP    WRITE1    WRITE SECTOR
 DE06 7E      DEEC        VERIFY  JMP    VRIFY1    VERIFY SECTOR
 DE09 7E      DEFB        RESTOR  JMP    RESTR1    RESTORE TO TRACK 0
 DE0C 7E      DF04        DRIVE   JMP    DRIVE1    SELECT DRIVE
 DE0F 7E      DF42        CHECK   JMP    CHECK1    CHECK DRIVE READY
 DE12 7E      DF4F        QCHECK  JMP    QCHEK1    QUICK CHECK READY
 DE15 7E      DE98        DINIT2  JMP    DINIT     NOT NEEDED
>DE18 7E      DE98        WARM    JMP    DINIT     NOT NEEDED
>DE1B 7E      DE5C        SEEK    JMP    SEEK1     SEEK TRACK

                          * VARIABLES
 DE1E 00                  CURDRV  FCB    0         CURRENT DRIVE NUMBER
 DE1F 1B                  CURSTP  FCB    $1B       CURRENT STEP SPEED
 DE20 00                  CURDNS  FCB    0         CURRENT DENSITY
 DE21 00                  CURSID  FCB    0         CURRENT SIDE


                          * Track Table stores head position (track)
                          * for each drive when it is not being used.
                          *
 DE22 00                  TRKTBL  FCB    0         0
 DE23 00                          FCB    0         1
 DE24 00                          FCB    0         2
 DE25 00                          FCB    0         3
```

```
                    * Drive step speed in msec (WD 1791)
                    *
                    * Code:    $18  $19  $1A  $18  SEEK, HLD, NO VRFY
                    * Five in.  6    12   20   30  MS
                    * Eight in. 3     6   10   15  MS
                    *
DE26 18             STPTBL  FCB     $18         0
DE27 18                     FCB     $18         1
DE28 18                     FCB     $18         2
DE29 18                     FCB     $18         3

                    * Density Table
                    *
                    * 00 - Single density, $FF - Double density
                    *
DE2A 00             DNSTBL  FCB     0           0
DE2B 00                     FCB     0           1
DE2C 00                     FCB     0           2
DE2D 00                     FCB     0           3


                    * READ SECTOR AT TRACK A, SECTOR B
                    *
DE2E 8D   2C        READ1   BSR     SEEK1
DE30 86   8C                LDA     #$8C        READ 1 SECTR. HD DLY, SIDE 0
DE32 8D   75                BSR     SETDC       SET CONTROLLER, DRV. REG.
DE34 8D   6A        READ2   BSR     SCHED       CHECK SCHEDULER
DE36 1A   10                SEI
DE38 B7   E018              STA     COMREG
DE3B 8D   5C                BSR     DELAY
DE3D C6   C0                LDB     #%11000000 SET DRQ, INTRQ MASK
DE3F 20   05                BRA     READ4

DE41 B6   E01B      READ3   LDA     DATREG      GET DATA
DE44 A7   80                STA     ,X+         STORE IT
DE46 F5   E014      READ4   BITB    DRVREG      GET DRQ, INTRQ
DE49 2B   F6                BMI     READ3       DRQ? YES, LOOP
DE4B 27   F9                BEQ     READ4       INTRQ? NO, LOOP

DE4D F6   E018              LDB     COMREG      GET STATUS
DE50 C5   10                BITB    #$10        RNF ERROR?
DE52 27   03                BEQ     READ5       NO
DE54 73   DE20              COM     CURDNS      YES, FLIP DENSITY
DE57 C5   1C        READ5   BITB    #$1C        SET RNF, CRC, LOST DATA
DE59 1C   EF                CLI
DE5B 39                     RTS
                    * SEEK TRACK A SECTOR B AND SET SIDE FLAG
                    *
DE5C F7   E01A      SEEK1   STB     SECREG      SET SECTOR
DE5F 8D   38                BSR     DELAY
DE61 4D                     TSTA                TRACK 0?
DE62 27   05                BEQ     SEEK2       YES, USE SD SECTOR COUNT
DE64 7D   DE20              TST     CURDNS      DOUBLE DENSITY?
DE67 26   06                BNE     SEEK3       YES, USE DOUBLE DENSITY
                                                                 COUNT
                    * USE SINGLE DENSITY COUNT
DE69 C1   0A        SEEK2   CMPB    #10         NO, SECTOR OVER 10?
DE6B 23   0D                BLS     SETS0       NO, SET SIDE 0
DE6D 20   04                BRA     SETS1       YES, SET SIDE 1

                    * USE DOUBLE DENSITY COUNT
DE6F C1   10        SEEK3   CMPB    #16         SECTOR OVER 16?
DE71 23   07                BLS     SETS0       NO, SET SIDE 0

DE73 C6   FF        SETS1   LDB     #$FF        SET SIDE 1 FLAG
DE75 F7   DE21              STB     CURSID
DE78 20   03                BRA     SEEK6

DE7A 7F   DE21      SETS0   CLR     CURSID      SET SIDE 0

DE7D B1   E019      SEEK6   CMPA    TRKREG      SAME AS BEFORE?
DE80 27   16                BEQ     DINIT       YES, EXIT
DE82 B7   E01B              STA     DATREG      PUT TRACK IN FDC
DE85 8D   12                BSR     DELAY
DE87 B6   DE1F              LDA     CURSTP      GET SEEK COMMAND


                    * DO COMMAND
                    *
DE8A B7   E018      DOCOM   STA     COMREG
DE8D 8D   0A                BSR     DELAY       DELAY AND WAIT


                    * WAIT UNTIL READY
                    *
DE8F 8D   0F        WAIT    BSR     SCHED       CHECK SCHEDULER
DE91 F6   E018              LDB     COMREG
DE94 C5   01                BITB    #1          BUSY?
DE96 26   F7                BNE     WAIT        YES
DE98 39             DINIT   RTS
```

```
                    * WASTE TIME FOR COMMAND
                    *
DE99 8D   00        DELAY   BSR     DELAY2
DE9B 8D   00        DELAY2  BSR     DELAY3
DE9D 8D   00        DELAY3  BSR     DELAY4
DE9F 39             DELAY4  RTS


                    * CHECK FLEX SPOOLER
                    *
DEA0 7D   CC34      SCHED   TST     FBFLAG      SPOOLING?
DEA3 27   03                BEQ     SCHED9      NO, EXIT
DEA5 113F                   SWI3
DEA7 12                     NOP
DEA8 39             SCHED9  RTS


                    * SET DISK CONTROLLER ROUTINE
                    *
                    *  IN: A - COMMAND
                    *      CURSID AND CURDNS SET
                    * OUT: A - COMMAND UPDATED, IF NEC.
                    *      DRIVE REG. SET FOR SIDE AND DENSITY
                    *      NOTE:  This version assumes that the
                    *      Side Byte on the disk is always set
                    *      to $00.  If your disks are formatted
                    *      with an honest side code, insert the
                    *      code marked %%%.
                    *
DEA9 F6   DE1E      SETDC   LDB     CURDRV      GET DRIVE NO.
DEAC 7D   DE21              TST     CURSID      SIDE 1?
DEAF 27   02                BEQ     SETDC1      NO, SKIP
DEB1 CA   40                ORB     #SIDBIT     SET SIDE 1 IN DRV. REG.
                    * ORA #SSOBIT SET SIDE 1N CMD REG. %%%
DEB3 7D   DE20      SETDC1  TST     CURDNS      DBL. DENS.?
DEB6 27   02                BEQ     SETDC2      NO, SKIP
DEB8 CA   80                ORB     #DNSBIT     SET DD
DEBA F7   E014      SETDC2  STB     DRVREG      TELL CONTROLLER
DEBD 39                     RTS


                    * WRITE SECTOR TO TRACK A, SECTOR B
                    *
DEBE 8D   9C        WRITE1  BSR     SEEK1
DEC0 86   AC                LDA     #$AC        WRITE 1 SCTR, HD DLY, SIDE 0.
DEC2 8D   E5                BSR     SETDC       SET CONTROLLER AND DRV. REG.
DEC4 8D   DA        WRITE2  BSR     SCHED       CHECK SCHEDULER
DEC6 1A   10                SEI
DEC8 B7   E018              STA     COMREG      WRITE COMMAND
DECB 8D   CC                BSR     DELAY
DECD C6   C0                LDB     #%11000000 SET DRQ, INTRQ MASK
DECF 20   03                BRA     WRITE4      WRITE DATA

DED1 B7   E01B      WRITE3  STA     DATREG      WRITE CHAR
DED4 A6   80        WRITE4  LDA     ,X+         GET CHAR FROM FCB
DED6 F5   E014      WRITE5  BITB    DRVREG      GET DRQ, INTRQ
DED9 2B   F6                BMI     WRITE3      DRQ? YES, LOOP
DEDB 27   F9                BEQ     WRITE5      INTRQ? NO, LOOP

DEDD F6   E018              LDB     COMREG      GET STATUS
DEE0 C5   10                BITB    #$10        RNF ERROR?
DEE2 27   03                BEQ     WRITE6      NO
DEE4 73   DE20              COM     CURDNS      YES, FLIP DENSITY
DEE7 C5   5C        WRITE6  BITB    #$5C        SET WP, RNF, CRC, LST
DEE9 1C   EF                CLI                                    DTA ERR.
DEEB 39                     RTS


                    * VERIFY LAST SECTOR WITH DUMMY READ
                    *
DEEC 86   88        VRIFY1  LDA     #$88        READ 1 SCTR, NO DLY, SIDE
DEEE 8D   B9                BSR     SETDC       SET CONTROLLER          0
DEF0 8D   AE        VRIFY2  BSR     SCHED       CHECK SCHEDULER
DEF2 1A   10                SEI
DEF4 8D   94                BSR     DOCOM
DEF6 C5   18                BITB    #$18        SET RNF, CRC ERRORS
DEF8 1C   EF                CLI
DEFA 39                     RTS


                    * RESTORE TO TRACK ZERO, SIDE ZERO
                    *
DEFB 8D   07        RESTR1  BSR     DRIVE1      SELECT DRIVE
DEFD 86   0B                LDA     #$B         RESTORE, NO HLD, NO VRFY,
DEFF 8D   89                BSR     DOCOM                             30MS
DF01 C5   D8                BITB    #$D8        SET ERRORS
DF03 39                     RTS
```

```
                           * SELECT DRIVE FROM FCB  AND SET SIDE ZERO
                           *
DF04 34    10              DRIVE1  PSHS   X
DF06 A6    03                      LDA    3,X        GET DRIVE NO. FROM FCB
DF08 81    03                      CMPA   #3         OVER 3?
DF0A 22    29                      BHI    OVER3      NO
DF0C 8D    2C                      BSR    FNDTRK     POINT AT OLD TRK IN TABLE
DF0E F6    E019                    LDB    TRKREG     GET TRACK
DF11 E7    84                      STB    0,X        SAVE IT IN TRKTBL
DF13 F6    DE20                    LDB    CURDNS     GET DRIVE DENSITY
DF16 E7    08                      STB    8,X        SAVE IT
DF18 B7    E014                    STA    DRVREG     SET NEW TRACK
DF1B B7    DE1E                    STA    CURDRV     SAVE A COPY
DF1E 8D    1A                      BSR    FNDTRK     POINT AT NEW DRIVE IN TBL
DF20 A6    04                      LDA    4,X        GET DRIVE STEP SPEED
DF22 B7    DE1F                    STA    CURSTP
DF25 A6    08                      LDA    8,X        GET DRIVE DENSITY
DF27 B7    DE20                    STA    CURDNS     SAVE IT
DF2A A6    84                      LDA    0,X        GET NEW DRIVE'S OLD TRACK
DF2C B7    E019                    STA    TRKREG     PUT IN CONTROLLER
DF2F BD    DE99                    JSR    DELAY
DF32 5F                            CLRB              NO ERROR
DF33 35    90              DRIVE9  PULS   X,PC

DF35 53                    OVER3   COMB              SET CARRY
DF36 C6    0F                      LDB    #$0F       LOAD ERROR CODE
DF38 20    F9                      BRA    DRIVE9     EXIT

DF3A 8E    DE22            FNDTRK  LDX    #TRKTBL
DF3D F6    DE1E                    LDB    CURDRV     GET DRIVE NO
DF40 3A                            ABX               POINT AT TABLE ENTRY
DF41 39                            RTS


                           * CHECK DRIVE READY
                           *
DF42 8D    0B              CHECK1  BSR    QCHEK1     CHECK DRIVE NO.
DF44 25    08                      BCS    ISRDY9     NOT READY, EXIT
DF46 8E    FFFF                    LDX    #$FFFF     DELAY 500 MS (1MHZ)
DF49 30    1F              CHECK2  LEAX   -1,X
DF4B 26    FC                      BNE    CHECK2     DONE?

DF4D 5F                    ISRDY   CLRB              CLEAR CARRY
DF4E 39                    ISRDY9  RTS


                           * QUICK CHECK DRIVE READY
                           *
DF4F B6    E018            QCHEK1  LDA    COMREG     TURN ON DRIVES
DF52 A6    03                      LDA    3,X        GET DRIVE NUMBER
DF54 81    01                      CMPA   #MAXDRV    LESS OR EQUAL?
DF56 23    F5                      BLS    ISRDY      YES

DF58 53                    NOTRDY  COMB              SET CARRY
DF59 C6    80                      LDB    #$80       SET ERROR CODE
DF5B 39                            RTS


                           * VERSION CODE
                           *
DF5C 20 46 44 52                   FCC    / FDRVPT69 5-4-85/
DF60 56 50 54 36
DF64 39 20 35 2D
DF68 34 2D 38 35
                   DF6C  DRVEND  EQU    *

                                 END

0 ERROR(S) DETECTED

SYMBOL TABLE:

CHECK   DE0F   CHECK1  DF42   CHECK2  DF49   COLDS   CD00   COMREG  E018
CURDNS  DE20   CURDRV  DE1E   CURSID  DE21   CURSTP  DE1F   DATREG  E01B
DELAY   DE99   DELAY2  DE9B   DELAY3  DE9D   DELAY4  DE9F   DINIT   DE98
DINIT2  DE15   DNSBIT  0080   DNSTBL  DE2A   DOCOM   DEBA   DRIVE   DE0C
DRIVE1  DF04   DRIVE9  DF33   DRVEND  DF6C   DRVREG  E014   FBFLAG  CC34
FNDTRK  DF3A   ISRDY   DF4D   ISRDY9  DF4E   MAXDRV  0001   NOTRDY  DF58
OVER3   DF35   QCHECK  DE12   QCHEK1  DF4F   READ    DE00   READ1   DE2E
READ2   DE34   READ3   DE41   READ4   DE46   READ5   DE57   RESTOR  DE09
RESTR1  DEFB   SCHED   DEAD   SCHED9  DEAB   SECREG  E01A   SEEK    DE1B
SEEK1   DE5C   SEEK2   DE69   SEEK3   DE6F   SEEK6   DE7D   SETDC   DEA9
SETDC1  DEB3   SETDC2  DEBA   SETS0   DE7A   SETS1   DE73   SIDBIT  0040
SSOBIT  0002   STPTBL  DE26   TRKREG  E019   TRKTBL  DE22   VERIFY  DE06
VRIFY1  DEEC   VRIFY2  DEFD   WAIT    DEBF   WARM    DE18   WRITE   DE03
WRITE1  DEBE   WRITE2  DEC4   WRITE3  DED1   WRITE4  DED4   WRITE5  DED6
WRITE6  DEE7
```

REVISED PT-69 SSO & DDEN LOGIC



*** CLASSIFIED ADVERTISING ***

NEW PRODUCT ANNOUNCEMENT

Introl Corporation
647 W. Virginia St.
Milwaukee, WI 53204
(414) 276-2937

or

Artisan Systems Corp.
PO Box 253
Revere, MA 02151
(617) 846-8323

Subject: INT OL INTRODUCES INOS OPERATING SYSTEM FOR THE 6809-BASED ARTISAN DP-09 COMPUTER

Introl Corporation is pleased to announce the release of the Introl INOS Operating System for the Artisan Systems Corporation 6809-based DP-09 Single Board Computer.

INOS is a high-performance, multiuser, multitasking operating system for 6809-based computers that is very similar to the UNIX operating system. In combination with the DP-09, which is ideally suited to running INOS, the INOS/DP-09 System proves to be a particularly powerful and effective computer system for 1 to 4 users. During the past one and one-half years, in fact, Introl has been using INOS/DP-09 systems exclusively for all of its own in-house software development work and INOS/DP-09 has consistently demonstated itself to be an exceptionally useful, efficient, and reliable development system throughout that period of time.

In conjunction with the Introl-C compiler, the Introl Standalone Development System, and the various Macro Cross Assemblers available from Introl for INOS, INOS/DP-09 is very well suited for a variety of software development activities, including program development for UNIX/INOS environments as well as for stand-alone ROM-based environments. INOS/DP-09 is also a capable multi-user business/generel purpose system for high performance word and date processing.

INOS/DP-09 facilitates the development of future applications in a UNIX-like environment while mainteining compatebility with the past, incorporating a UniFlex compatability mode that allows most UniFlex applications programs - such as Stylogreph, Dynecalc, Sculpture, etc - to execute under INOS without modification.

INOS is a high performance software system that has been designed with the future in mind, being written in C in a highly portable manner to allow its future migration to processors such as the 68000 end 32000. INOS is also optionally available in several configurable versions to enable the addition of hardware device drivers and even to allow INOS to be ported to other 6809-based hardware configurations.

INOS FEATURES:

- Comprehensive utility command set (over 60)
- Multi-user, multi-tasking environment
- Tree structured directories
- Input/Output redirection and "pipes"
- Up to 6 simultaneous users on the DP-09
- Compatability mode to run UniFLEX programs
- UNIX Source code compatability
- Full support for the DP-09 hardware
- Makes the 6809 into a "supermicro"

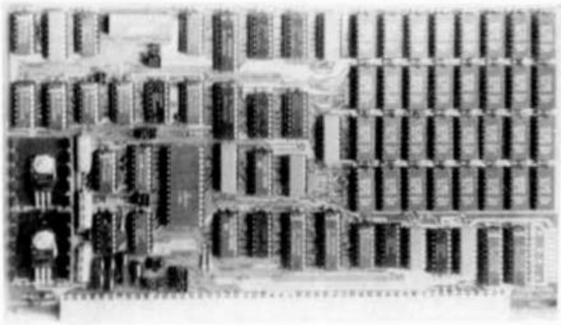
A partial listing of utilities standard under INOS:

calendar	cat	chmod	cmp
cp	csh	cu	date
df	diff	ed	expand
gets	grep	hd	iroff
kill	ln	login	ls
mail	mesg	mkdir	mv
ned	newgrp	od	passwd
printenv	ps	pwd	rm
size	sleep	sort	split
stty	su	sum	sync
tee	touch	tty	unexpand
uniq	wc	who	write

DP-09 FEATURES:

- 512K bytes memory standard (expandable to 768K)
- Up to 192K bytes of EPROM
- Dual 2 MZ 6809 processors (with one CPU dedicated to handling disk I/O)
- Six RS-232 Serial ports (with software selectable baudrates)
- 3.5, 5, & 8 inch floppy interface for up to four floppies
- SCSI interface for winchester disks
- Centronics-compatible parallel port

As an introduction to INOS/DP-09, Introl and Artisan are joining forces and, for a limited time, will be offering a starter system - consisting of the INOS operating system, the Artisan DP-09, and the Introl-C/6809 compiler - at the low introductory price of only $1495. Simply add a terminal, power supply, floppy disks, and optional winchester disk and your system is complete. Fully pre-assembled systems will be available soon from Artisan Systems for under $4000. OEM licensing is available for both INOS and the DP-09.



INDUSTRIAL PASCAL FOR THE 68000

If you're looking for a language to write real-time process control software, look no further. With the rising cost of labor, it is becoming critical that a high level language be used whenever possible. Find out why over 1400 companies have switched to OmegaSoft Pascal for their demanding applications.

OmegaSoft Pascal takes the Pascal framework and expands the basic data types, operators, functions, and memory allocation to fit the needs of real-time systems. These additions fit in the same structure as Pascal and enhance its usefulness without impairing the excellent readability, ease of maintenance, and structured design.

The compiler generates assembly language for assembly and link to run on the target system. Since a true relocating assembler and linking loader is used, only those runtime modules required are automatically linked in, providing a smaller object module than other compilers.

Large Pascal programs can be split up into conveniently sized modules to speed the development process. Procedures, functions, and variables can be referenced between Pascal modules and assembly language modules by using Pascal directives.

The compiler package includes an interactive, symbolic debugger. The debugger allows setting of breakpoints, displaying and changing variables, and tracing statements. Debugging can also be done at the assembly language level when needed. The debugger allows very fast turnaround for programs to be run on the host system (target system debugger coming soon).

The compiler package also includes a full relocatable macro assembler and linking loader. These are designed to support the compiler but may also be used for general assembly language development. in addition, a full screen editor is included which can be used with a variety of intelligent terminals.

Full source code is included for the runtime library, the debugger, the screen editor, and other support utilities.

Versions to run under the OS-9/68000 and VERSAdos operating systems are currently available to end-users and OEM's. End user price is $900 (domestic) or $925 (international). A version for CP/M-68K is available for OEM use, with OEM versions for UNIX type operating systems to follow.

Similar products to run on a 6809 system and generate 6809 code are also available for most major 6809 operating systems.

T.M. OmegaSoft is a trademark of Certified Software Corporation. OS-9/68000 is a trademark of Microware. VERSAdos is a trademark of Motorola. CP/M-68K is a trademark of DRI. UNIX is a trademark of Bell Labs.

CERTIFIED SOFTWARE CORPORATION

616 Camino Caballo, Nipomo, CA 93444
Telephone: (805) 349-0202; Telex: 467013

GOOD NEWS!

C for the 6809 WAS NEVER BETTER!

INTROL-C/6809,Version1.5

Introl's highly acclaimed 6809 C compilers and cross-compilers are now more powerful than ever!

We've incorporated a totally new 6809 Relocating Assembler, Linker and Loader. Initializer support has been added, leaving only bitfield-type structure members and doubles lacking from a 100% full K&R implementation. The Runtime Library has been expanded and the Library Manager is even more versatile and convenient to use. Best of all, compiled code is just as compact and fast-executing as ever - and even a bit more so! A compatible macro assembler, as well as source for the full Runtime Library, are available as extra-cost options.

Resident compilers are available under **Uniflex, Flex** and **OS9.**

Cross-compilers are available for **PDP-11/UNIX** and **IBM PC/PC DOS** hosts.

Trademarks:
Introl-C, Introl Corporation
Flex and Uniflex, Technical Systems Consultants
OS9, Microware Systems
PDP-11, Digital Equipment Corp.
UNIX, Bell Laboratories
IBM PC, International Business Machines

For further information, please call or write.

INTROL
CORPORATION

647 W. Virginia St.
Milwaukee, WI 53204
(414) 276-2937

COMPILER EVALUATION SERVICES
By: Ron Anderson

The S.E. MEDIA Division of Computer Publishibg Inc., is offering the following SUBSCRIBER SERVICE:

COMPILER COMPARISON AND EVALUATION REPORT

Due to the constant and rapid updating and enhancement of numerous compilers, and the different utility, appeal, speed, level of communication, memory usage, etc., of different compilers, the following services are now being offered with periodic updates.

This service, with updates, will allow you who are wary or confused by the various claims of compiler vendors, an opportunity to review comparisons, comments, benchmarks, etc., concerning the many different compilers on the market, for the 6809 microcomputer. Thus the savings could far offset the small cost of this service.

Many have purchased compilers and then discovered that the particular compiler purchased either is not the most efficient for their purposes or does not contain features necessary for their application. Thus the added expense of purchasing additional compiler(s) or not being able to fully utilize the advantages of high level language compilers becomes too expensive.

The following COMPILERS are reviewed initially, more will be reviewed, compared and benchmarked as they become available to the author:

PASCAL "C" GSPL WHIMSICAL PL/9

Initial Subscription - $ 39.95
(includes 1 year updates)
Updates for 1 year - $ 14.50

S.E. MEDIA - C.P.I.
5900 Cassandra Smith Rd.
Hixson, Tn. 37343
(615) 842-4601

OS-9™ SOFTWARE

SDISK—Standard disk driver module allows the use of 35, 40, or 80 track double sided drives with COCO OS-9 plus you can read/write/format the OS-9 formats used by other OS-9 systems. **$29.95**

SDISK+BOOTFIX—As above plus boot directly from a double sided diskette **$35.95**

FILTER KIT #1—Eleven OS-9 utilities for "wild card" directory lists, copies, moves, deletes, sorts, etc. Now includes disk sector edit utility also. **$29.95 ($31.95)**

FILTER KIT #2—Macgen command macro generator builds new commands by combining old ones with parameter substitution, 10 other utilities. **$29.95 ($31.95)**

HACKER'S KIT #1—Disassembler and related utilities allow disassembly from memory, file. **$24.95 ($26.95)**

PC-XFER UTILITIES—Utilities to read/write and format MS-DOS™ diskettes on CoCo under OS-9. Also transfer files between RS disk basic and OS-9 (requires sdisk). **$45.00**

BOLD prices are CoCo OS-9 format disk, other formats (in parenthesis) specify format and OS-9 level. All orders prepaid or COD, VISA and MasterCard accepted. Add $1.50 S&H on prepaid, COD actual charges added.

SS-50C
1 MEGABYTE RAM BOARD

Full megabyte of ram with disable options to suit any SS-50 6809 system. High reliability, can replace static ram for a fraction of the cost. $895 for 2 Mhz or $995 for 2.25 Mhz board assembled, tested and fully populated. (Add $6 shipping and insurance, quantity discounts available.)

D.P. Johnson, 7655 S.W. Cedarcrest St.
Portland, OR 97223 (503) 244-8152
(For best service call between 9-11 AM Pacific Time.)

OS-9 is a trademark of Microware and Motorola Inc.
MS-DOS is a trademark of Microsoft, Inc.

	ASSEMBLED	BARE
Elektra Super Bus (Fits Current Elektra Chassis)	Coming	
Elektra Super 680XX Boards	Coming	
Cabinet w/Power Supply (Largest SS-50 Cab. Available)	450.00	
Disk Regulator	50.00	
Disk Regulator (Heavy Duty)	75.00	
Motherboard (Gold Connectors)	395.00	
2MHz 6802/6809 CPU	275.00	65.00
Dual Port Serial Card	95.00	40.00
Quad Port Parallel Card	80.00	40.00
64K Static Ram Card	250.00	N/A
5"/8" 2MHz Super Floppy Controller (The Best)	295.00	100.00
Flex/Star-DOS Drivers	30.00	
OS-9 Drivers	50.00	
Elektra OS-9 w/Editor, Assembler, Debugger	250.00	
Elektra Star-DOS	75.00	

Write or phone for current pricing

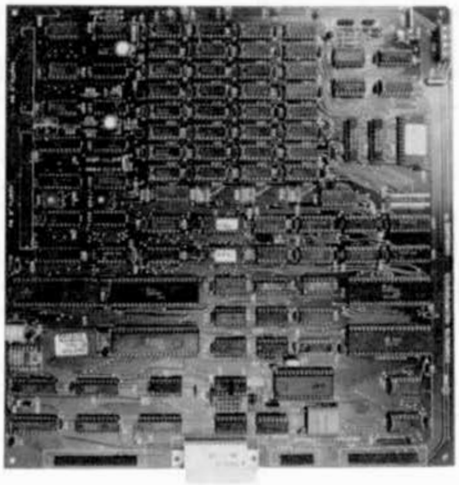
AAA Chicago Computer Center
120 Chestnut Lane — Wheeling IL 60090
(312) 459-0450

	ASSEMBLED	BARE
4K Humbug	75.00	
2K Microbug	40.00	
7-84 MBute Winchester Systems	-Phone-	
Removable Cartridge Winchester Systems	-Phone-	
DMA Winchester/Floppy SS-50 Host Interface	395.00	250.00
HD-5 Dual Floppy Cabinet with Power Supply	150.00	
HD-5W Floppy/Winchester Cabinet with Power Supply	199.00	
HD-5DH Dual Winchester Cabinet with Power Supply	250.00	
Ribbon Cable for Dual Outboard Floppies	40.00	
Ribbon Cable for Dual Inboard Floppies	35.00	
HD-8 Dual 8" Floppy Cabinet with Power Supply	350.00	
Ribbon Cable for Dual 8" Floppies	45.00	
Custom Cables On Request	-Phone-	
30 Pin Prototyping Board	30.00	
40 Pin Prototyping Board	50.00	
CONNECTORS		
Gold Plated 10 Pin Male (Square Posts) or Female	1.50	
Tin Plated 10 Pin Male (Square Posts) or Female	1.50	

Technical Consultation available most weekdays from 4 PM to 6 PM CST

ELEKTRA is a trademark of AAA Chicago Computer Center
FLEX is a trademark of Technical Systems Consultants, Inc.

OS-9 and Basic09 are trademarks of Microware Systems Corp.
UniFLEX is a registered trademark of Technical Systems Consultants, Inc.



WINDRUSH MICRO SYSTEMS

UPROM II

PROGRAMS and VERIFIES: I2758, T2508, I2716, T2516, I2732/2732A, M68764/6, I2764/2764A, T2564, I27128/27128A, and I27256. I=Intel, T=Texas, M=Motorola.

NO PERSONALITY MODULES REQUIRED!

TRI-VOLT EPROMS ARE NOT SUPPORTED

INTEL's intelligent programming (tm) implemented for Intel 2764, 27128 and 27256 devices. Intelligent programming reduces the average programming time of a 2764 from 7 minutes to 1 minute 15 seconds (under FLEX) with greatly improved reliability.

Fully enclosed pod with 5' of flat ribbon cable for connection to the host computer MC6821 PIA interface board.

MC6809 software for FLEX and OS9 (Level 1 or 2, Version 1.2).

BINARY DISK FILE offset loader supplied with FLEX, MDOS and OS9.

Menu driven software provides the following facilities:

```
a. FILL ........... a selected area of the buffer with a HEX char.
b. MOVE ........... blocks of data.
c. DUMP ........... the buffer in HEX and ASCII.
d. FIND ........... a string of bytes in the buffer.
e. EXAMINE/CHANGE . the contents of the buffer.
f. CRC ............ checksum a selected area of the buffer.
g. COPY ........... a selected area of an EPROM into the buffer.
h. VERIFY ......... a selected area of an EPROM against the buffer.
i. PROGRAM ........ a selected area of an EPROM with data in the buffer.
j. SELECT ......... a new EPROM type (return to types menu).
k. ENTER .......... the system monitor.
l  RETURN ......... to the operating system.
l. EXECUTE ........ any DOS utility (only in FLEX and OS9 versions).
```

FLEX AND OS9 VERSIONS AVAILABLE FROM GIMIX.   SSB/MDOS CONTACT US DIRECT.

PL/9

- Friendly inter-active environment where you have INSTANT access to the Editor, the Compiler, and the Trace-Debugger, which, amongst other things, can single step the program a SOURCE line at a time. You also have direct access to any FLEX utility and your system monitor.
- 375+ page manual organized as a tutorial with plenty of examples.
- Fast SINGLE PASS compiler produces 8K of COMPACT and FAST 6809 machine code output per minute with no run-time overheads or license fees.
- Fully compatible with TSC text editor format disk files.
- Signed and unsigned BYTES and INTEGERS, 32-bit floating point REALs.
- Vectors (single dimension arrays) and pointers are supported.

```
• Mathematical expressions: (+), (-), (*), (/), modulus (\), negation (-)
• Expression evaluators:    (=), (<>), (<), (>), (>=), (<=)
• Bit operators:            (AND), (OR), (EOR/XOR), (NOT), (SHIFT), (SWAP)
• Logical operators:        (.AND), (.OR), (.EOR/XOR)
```

- Control statements: IF..THEN..ELSE, IF..CASE1..CASE2..ELSE, BEGIN..END, WHILE.., REPEAT..UNTIL, REPEAT..FOREVER, CALL, JUMP, RETURN, BREAK, GOTO.
- Direct access to (ACCA), (ACCB), (ACCD), (XREG), (CCR) and (STACK).
- FULLY supports the MC6809 RESET, NMI, FIRQ, IRQ, SWI, SWI2, and SWI3 vectors. Writing a self-starting (from power-up) program that uses ANY, or ALL, of the MC6809 interrupts is an absolute snap!
- Machine code may be embedded in the program via the 'GEN' statement. This enables you to code critical routines in assembly language and embed them in the PL/9 program (see 'MACE' for details).
- Procedures may be passed and may return variables. This makes them functions which behave as though they were an integral part of PL/9.
- Several fully documented library procedure modules are supplied: IOSUBS, BITIO, HARDIO, HEXIO, FLEXIO, SCIPACK, STRSUBS, BASTRING, and REALCON.

'... THIS IS THE MOST EFFICIENT COMPILER I HAVE FOUND TO DATE.'

Quoted from Ron Andersons FLEX User Notes column in '68. Need we say more?

MACE/XMACE/ASM05

All of these products feature a highly productive environment where the editor and the assembler reside in memory together. Gone are the days of tedious disk load and save operations while you are debugging your code.

- Friendly inter-active environment where you have instant access to the Editor and the Assembler, FLEX utilities and your system monitor.
- MACE can also produce ASMPROCs (GEN statements) for PL/9 with the assembly language source passed to the output as comments.
- XMACE is a cross assembler for the 6800/1/2/3/8 and supports the extended menmonics of the 6303.
- ASM05 is a cross assembler for the 6805.

D-BUG

LOOKING for a single step tracer and mini in-line disassembler that is easy to use?? Look no further, you have found it. This package is ideal for those small assembly language program debugging sessions. D-BUG occupies less than 6K (including its stack and variables) and may be loaded anywhere in memory. All you do is LOAD IT, AIM IT and GO! (80 col VDUs only).

McCOSH 'C'

This is as complete a 'C' compiler as you will find on any operating system for the 6809. It is completely compatible with UNIX VII and only lacks 'bit-fields' (which are of little practical use in an 8-bit world!).

- Produces very efficient assembly language source output with the 'C' source optionally interleaved as comments.
- Built-in optimizer will shorten object code by about 11%.
- Supports interleaved assembly language programs.
- INCLUDES its own assembler. The TSC relocating assembler is only required if you want to generate your own libraries.
- The pre-processor, compiler, optimizer, assembler and loader all run independently or under the 'CC' executive. 'CC' makes compiling a program to executable object as simple as typing in 'CC,HELLO.C <RETURN>'.

IEEE-488

- SUPPORTS ALL PRINCIPAL MODES OF THE IEEE-488 (1975/8) BUS SPECIFICATION:

```
- Talker               - Serial Poll      - Single or Dual Primary Address
- Listener             - Parallel Poll    - Secondary Address
- System Controller    - Group Trigger    - Talk only ... Listen only
```

- Fully documented with a complete reprint of the KILOBAUD article on the IEEE bus and the Motorola publication 'Getting aboard the IEEE Bus'.
- Low level assembly language drivers suitable for 6800, 6801, 6802, 6803, 6808 and 6839 are supplied in the form of listings. A complete back to back test program is also supplied in the form of a listing. These drivers have been extensively tested and are GUARANTEED to work.
- Single S-30 board (4, 8 or 16 addresses per port), fully socketed, gold plated bus connectors and IEEE interface cable assembly.

PRICES

D-BUG	(6809 FLEX only)	$ 75.00
MACE	(6809 FLEX only)	$ 75.00
XMACE	(6809 FLEX only)	$ 98.00
ASM05	(6809 FLEX only)	$ 98.00
PL/9	(6809 FLEX only)	$198.00
'C'	(6809 FLEX only)	$295.00
IEEE-488	with IEEE-488 cable assembly	$298.00
UPROM-II/U	with one version of software (no cable or interface) ..	$395.00
UPROM-II/C	as above but complete with cable and S-30 interface	$545.00
CABLE	5' twist-n-flat 50 way cable with IDC connectors	$ 35.00
S-30 INT	SS-30 interface for UPROM-II	$130.00
EXOR INT	Motorola EXORbus (EXORciser) interface for UPROM-II ...	$195.00
UPROM SFT	Software drivers for 2nd operating system. Specify FLEX or OS9 AND disk size!	$ 35.00
UPROM SRC	Assembly language source (contact us direct)	

ALL PRICES INCLUDE AIR MAIL POSTAGE

Terms: CWO. Payment by Int'l Money Order. VISA or MASTER-CARD also accepted.

WORSTEAD LABORATORIES, NORTH WALSHAM,
NORFOLK, ENGLAND.   NR28 9SA.
TEL: 44 (692) 404086
TLX: 975548 WMICRO G

WE STOCK THE FOLLOWING COMPANIES PRODUCTS:
GIMIX, SSB, FHL, MICROWARE, TSC, LUCIDATA, LLOYD I/O,
& ALFORD & ASSOCIATES.

FLEX (tm) is a trademark of Technical Systems Consultants, OS-9 (tm) is a trademark of Microware Systems Corporation, MDOS (tm) and EXORciser (tm) are trademarks of Motorola Incorporated.

COMPARE

our EPROM PROGRAMMER with the field.

All data taken directly from manufacturer's current advertising. Software, interfaces, or personality modules may also be required at additional cost.

- • Triple voltage EPROM
- • Supplied in kit form

		A	B	C	D	E	F
INTERFACE	S30	PAR	PAR	SER	S30	SER	SER
INTELLIGENT	NO	N	NO	YES		YES	YES
PROGRAMS							
2704•			●				●
2508	●			●	●	●	
2708•			●		●		●
2758	●	●	●	●		●	●
2516	●	●	●	●	●	●	
2716	●	●	●	●	●	●	●
2716•			●		●		●
2532	●		●	●	●	●	●
2732	●		●	●	●	●	●
2732A	●		●		●	●	●
2564	●		●		●	●	●
2764	●		●		●	●	●
2528	●				●		
27128	●						
2816							●
68764			●				
8748						●	
8749						●	
TOTAL	11	3	12	6	11	11	11
PRICE	$125	$45•	$169	$289	$375	$489	$575

OPAM EPROM Programmer, $125. Personality module for 2508, 2758, 2516, and 2716 included. Specify CPU, disk size, and operating system (TSC's FLEX or SSB's DOS) when ordering. Manual only, $10; refundable with OPAM purchase.

UNITEK • P.O. Box 671 • Emporia, VA 23847

TMP SOFTWARE ANNOUNCES

****

A Popular FLEX Database Program, DATAMAN Is being Re-released!

****

$195.00

POWERFUL CAPABILITIES:

- Each field can contain up to 126 characters, and each field name can contain up to 29 characters.
- Field types can be alpha, numeric or monetary.
- **DATAMAN** lets you produce vertical or horizontal reports of your database, or create customized letters and forms to pull data from a database using **DATAMAN** files processed by the TSC Text Processor.
- **DATAMAN** allows you to look up records quickly, to merge unlike databases and to move records from one database into another.
- **DATAMAN** includes a Label Generator to produce mailing, shipping or inventory labels.
- To sort records based on the geographic, historical, name, or monetary criteria you select, **DATAMAN** creates Sort Files which can then be used with the TSC Sort Merge package.

GIVE YOU MORE ABILITIES:

Our Users put **DATAMAN** to work for them to do customer mailings, past due notices, real estate listings, invoicing, sales analysis, activity scheduling, inventories, employee records and client profile reports.

Also included in this package is **DATARAND**, the extension program to **DATAMAN** which enhances **DATAMAN** programs.

DATAMAN runs under TSC Extended Basic. The TSC Sort Merge and Text Processor programs are also recommended. The Source Code for DATAMAN and DATARAND is included on the release disks.

ORDERING INFORMATION: TMP SOFTWARE
2431 E. Douglas • Wichita, KS • 67211
OR CALL TOLL FREE: 1-800-255-1382 Ext. 47
We accept VISA, MC, AMEX, money orders and checks.

NOTE: TMP Software also offers the TMP POWER MANAGER for OS 9 systems, and the TMP FREEFORM FILER for OS 9 and Uniflex systems.
Flex and TSC are trademarks of Technical Systems Consultants, Inc.

68' MICRO JOURNAL

Disk-1 Filesort, Minicat, Minicopy, Minifms, **Lifetime, **Poetry, **Foodlist, **Diet.

Disk-2 Diskedit w/ inst.& fixes, Prime, *Prmod **Snoopy, **Football, **Hexpawn, **Lifetime

Disk-3 Cbug09, Sec1, Sec2, Find, Table2, Intext, Disk-exp, *Disksave.

Disk-4 Mailing Program, *Finddat, *Change, *Testdisk.

DISK-5 *DISKFIX 1, *DISKFIX 2, **LETTER, **LOVESIGN **BLACKJAK, **BOWLING.

Disk-6 **Purchase Order, Index (Disk file indx)

Disk-7 Linking Loader, Rload, Harkness

Disk-8 Crtest, Lanpher (May 82)

Disk-9 Datecopy, Diskfix9 (Aug 82)

Disk-10 Home Accounting (July 82)

Disk-11 Dissembler (June 84)

Disk-12 Modem68 (May 84)

Disk-13 *Initmf68, Testmf68, *Cleanup, *Dskalign, Help, Date.Txt

Disk-14 *Init, *Test, *Terminal, *Find, *Diskedit, Init.Lib

Disk-15 Modem9 + Updates (Dec. 84 Gilchrist) to Modem9 (April 84 Commo)

Disk-16 Copy.Txt, Copy.Doc, Cat.Txt, Cat.Doc

Disk-17 Match Utility, RATBAS (A Basic Preprocessor) June 85

Disk-18 Parse.Mod, Size.Cmd (Sept. 85 Armstrong), CMDCODE, CMD.Txt (Sept. 85 Spray)

Disk-19 Clock, Date, Copy, Cat, PDEL.Asm & Doc., Errors.Sys, Do, Log.Asm & Doc.

Disk-20 UNIX Like Tools (July & Sept. 85 Taylor & Gilchrist). Dragon.C, Grep.C, LS.C, FDUMP.C

NOTE:

This is a reader service ONLY! No Warranty is offered or implied, they are as received by '68' Micro Journal, and are for reader convenience ONLY (some MAY include fixes or patches). Also 6800 and 6809 programs are mixed, as each is fairly simple (mostly) to convert to the other.

PRICE: 8" Disk **$14.95** - 5" Disk **$12.95**

68' Micro Journal
5900 Cassandra Smith Rd.
Hixson, Tn. 37343
(615)842-4600

* Indicates 6800
** Indicates BASIC SWTPC or TSC
6809 no Indicator.

MASTER CARD - VISA Accepted
Foreign -- add **10%** for Surface
or **20%** for Air!!

TRS-80+ MOD I, III, COCO, TI99/4a
TIMEX 1000, OSBORNE, others

GOLD PLUG - 80

Eliminate disk reboots and data loss due to oxidized contacts at the card edge connectors.
GOLD PLUG 80 solders to the board edge connector. Use your existing cables. (if gold plated)

SPECIAL PRICES

GOLD PLUG 80 Mod I (6)	$44.95	~~$54.95~~
Keyboard/EI (mod I)	15.95	~~18.95~~
Individual connectors	7.95	~~9.95~~
COCO Disk Module (2)	16.95	~~18.95~~
Ground tab extensions	INCL	~~1.00~~
Disk Drives (all R.S.)	7.95	~~9.95~~
Gold Disk Cable 2 Drive		29.95
Four Drive Cable		39.95
GOLD PLUG 80 Mod III (6)		54.95
Internal 2 Drive Cable		29.95
Mod III Expansion port		10.95
USA shipping $1.45		Can/Mex $4.
Foreign $7.		TEXAS 5% TAX

COCO MODULE INSTALLATION AVAILABLE

Ask your favorite dealer or order direct

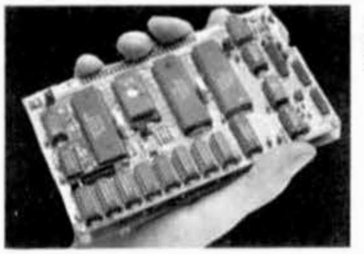
ORDER TODAY!

E.A.P. CO.
P.O. BOX 14
KELLER, TEXAS 76248
(817) 498-4242

VISA

MC/VISA

+ trademark Tandy Corp



Available Assembled and Tested

Compact Flexible 6809 Computer

The new ST-2900 system — a complete 64K small business or hobbyist computer is only one of its many possible configurations. Among its features are:

- Small enough to hold in your hand! (Eurocard size: 3.9" x 6.3")
- Two board "system" for greater versatility than single board computers.
- CPU Board — powerful 6809E processor, 16K or 64K RAM, 1K-32K EPROM, 2 RS232 serial ports with software programmable baud rates, 16 bit counter/timer. Run the CPU board all by itself, or plug your own custom board or our FDC board into the expansion connector.
- FDC Board — double-sided/double-density floppy disk controller with adjustment free digital data separator and write precompensation, 2 8-bit parallel ports, 2 16-bit counter/timers, prototyping area.
- Available as bare PC boards or **fully assembled and tested** boards. All have solder mask both sides plus silkscreened component overlay.
- **OS-9 for only $49?** Well, not quite. But that's all you pay for our OS-9 Conversion Package which lets you use the low cost Radio Shack CoCo version of OS-9 on our ST-2900 system. Save $131 off the suggested list price of OS-9! **No programming is involved.** Supports CoCo OS-9 and standard OS-9 disk formats.

		OS-9 Conversion Package	$49
• CPU bare board plus EPROM	$45	FLEX Conversion Package	$29
FDC bare board	$38	CPU + FDC + OS-9 Conversion	$119
CPU + FDC board set assembled and tested			$329

- Add $5 shipping/handling ($10 overseas). These prices are in U.S. funds. Canadian orders: call or write for prices. Terms: money order, certified check, VISA.

Coming soon: 512K RAM expansion!

(FLEX is a trademark of Technical Systems Consultants. OS-9 is a trademark of Microware and Motorola)

SARDIS TECHNOLOGIES

Write for free brochure and complete price list.
(604) 255-4485 (4-5 pm PST)

2261 E. 11th Ave. Vancouver, B.C., Canada V5N 1Z7

LLOYD I/O™
Computer Engineers

19535 NE GLISAN • PORTLAND, OR 97230 (USA)
PHONE: (503) 666-1097 • TELEX: 910 380 5448 LLOYD I O

K-BASIC™ *IS HERE*

K-BASIC is a TSC XBASIC (XPC) compatible COMPILER
for OS9 & FLEX...price $199

Here at last is a compiler for BASIC that will compile all your XBASIC programs. K-BASIC compiles TSC's XBASIC and XPC programs to machine code. K-BASIC is ready now to save you money and time by teaching your computer to:
• Think Faster • Conserve Memory • Be Friendlier

Call (503) 666-1097 for our CATALOG.
We have many programs for serious software developers!

DO™

Micro BASIC for OS9...$149

A structured mirco BASIC for general system control featuring: Parameter passing, 10 string variables, 26 numeric variables, subroutines, nested loops, interactive I/O, sequential files, and time variables (for applications executing in the background required to execute procedures such as disk or file backups.) Includes the SEARCH and RESCUE UTILITIES™. (For OS9 ONLY.)

SEARCH and RESCUE UTILITIES™

for OS9...$35

A super directory search utility. Output may be piped to the included utilities to perform file: COPIES, DELETES, MOVES, LISTING (pagination), and FILTERING. Some filtering utility programs are included: of interest is the FILE DATE CHECKING utilities: YOUNGER and DRAFT (Level 2). (For OS9 Level 1 and 2.)

PATCH™

Modem Communications for OS9...$39

PATCH is a modem communications program for OS9 featuring: KEY MACROS, ASCII TEXT AND BINARY FILE UP/DOWN LOADING, PRINTER COPY, and HELP MENUS. We use it several times each day with our TELEX service. PATCH is convenient and easy to use. Key macros may be pre-stored and loaded at any time.

CRASMB™

CROSS ASSEMBLER PACKAGE
for OS9 & FLEX... all for $399
Motorola CPU's...$150
Intel CPU's...$150. Others...$150

CRASMB is the highly acclaimed cross assembler package for OS9 and FLEX systems. It turns your 6809 computer into a development station for these target CPUs:

6800 6801 6804 6805 6809 6811 6502
7000 1802 8048 8051 8080 8085 Z8 Z80
(68000 16/32 bit cross assembler...$249)

CRASMB features: Macros, Conditionals, Long symbol names, Symbol cross reference tables. Object code in 4 formats (OS9, FLEX, S-1-S9, INTEL HEX).

VISA, MC, COD, CHECKS, ACCEPTED

USA:	LLOYD I/O (503 666 1097),	S.E. MEDIA (800 338 6800)
England:	Vivaway (0582 423425),	Windrush (0692 405189)
Germany:	Zacher Computer (65 25 299),	Kell Software (06203 6741)
Australia:	Paris Radio Electronics (344 9111)	

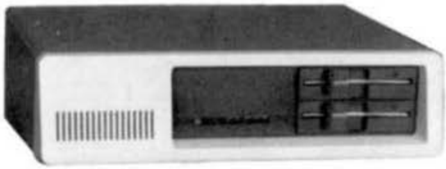
K-BASIC, DO, SEARCH and RESCUE UTILITIES,
PATCH, CRASMB and CRASMB 16.32 are trademarks of LLOYD I/O
OS9 is a ™ of Microware, FLEX is a ™ of TSC

PT-69 SINGLE BOARD COMPUTER SYSTEMS

NOW WITH WINCHESTER OR FLOPPY DISK DRIVES

The proven PT-69 Single Board Computer line is expanding! Systems now can be Winchester or floppy-based. Available also in a smaller cabinet without drives for dedicated systems with no mass storage requirements.

- 1 MHZ 6809E Processor
- Time-of-Day Clock
- 2 RS 232 Serial Ports (6850)
- 56K RAM 2K/4K EPROM
- 2 8-bit Parallel Ports (6821)
- 2797 Floppy Disk Controller



Winchester System

Floppy System

Custom Design Inquiries Welcome

• PT69XT WINCHESTER SYSTEM Includes 5 MEG Winchester Drive, 1 40-track DS/DD Drive, Parallel Printer Interface • choice of OS/9 or STAR-DOS	$1895.95
• PT69S2 FLOPPY SYSTEM Includes PT69 Board, 2 DS/DD 40-TRK 5¼" drives, cabinet, switching power supply, OS/9 or STAR-DOS	$ 949.95
• PT-69A ASSEMBLED & TESTED BOARD	$289
• OS/9	$200
• STAR-DOS	$ 50

CALL OR WRITE FOR ADDITIONAL CONFIGURATIONS

PERIPHERAL TECHNOLOGY

1480 Terrell Mill Rd., Suite 870
Marietta, Georgia 30067
Telex #880584

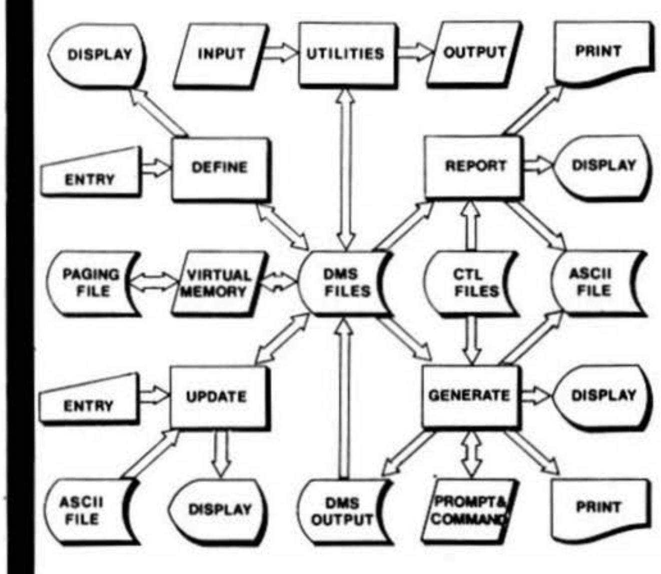
VISA/MASTERCARD/CHECK/COD

404/973-0042

*OS/9 is a trademark of Microware and Motorola

XDMS

Data Management System



System Architecture

WESTCHESTER Applied Business Systems
Post Office Box 187
Briarcliff Manor, N.Y. 10510

XDMS Data Management System

The XDMS Data Management System is available in three levels. Each level includes the XDMS nucleus, VMGEN utility and System Documentation for level III. XDMS is one of the most powerful systems available for 6809 computers and may be used for a wide variety of applications. XDMS users are registered in our database to permit distribution of product announcements and validation of user upgrades and maintenance requests.

XDMS Level I

XDMS Level I consists of DEFINE, UPDATE and REPORT facilities. This level is intended as an "entry level" system, and permits entry nd reporting of data on a "tabular" basis. The REPORT facility supports record and field selection, field merge, sorting, line calculations, column totals and report titling. Control is via a English-like language which is upward compatible with level II. XDMS Level I $129.95

XDMS Level II

Level II adds to Level I the powerful GENERATE facility. This facility can be thought of as a general file processor which can produce reports, forms and form letters as well as file output which may be re-input to the facility. GENERATE may be used in complex processing applications and is controlled by a English-like command language which encompasses that used by Level I. XDMS Level II $199.95

XDMS Level III

Level III includes all of level II plus a set of useful DMS Utilities. These utilities are designed to aid in the development and maintenance of user applications and permit modification of XDMS system parameters, input and output of XDMS files, display and modification of file format, graphic display of numerical data and other functions. Level III is intended for advanced XDMS users. XDMS Level III $269.95
XDMS System Documentation only ($10. credit toward purchase) . . $ 24.95

XACC Accounting System

The XACC General Accounting System is designed for small business environments of up to 50,000 accounts and inventory items. The system integrates accounting functions and inventory plus the general ledger, accounts receivable and payable functions normally sold separately in other systems. Features user defined accounts, products (or services), transactions, invoicing, etc. Easily configured to most environments.
XACC General Accounting System (Requires XDMS, pref. Lv. III) . . $299.95
XACC System Documentation only ($10. credit toward purchase) . . $ 24.95

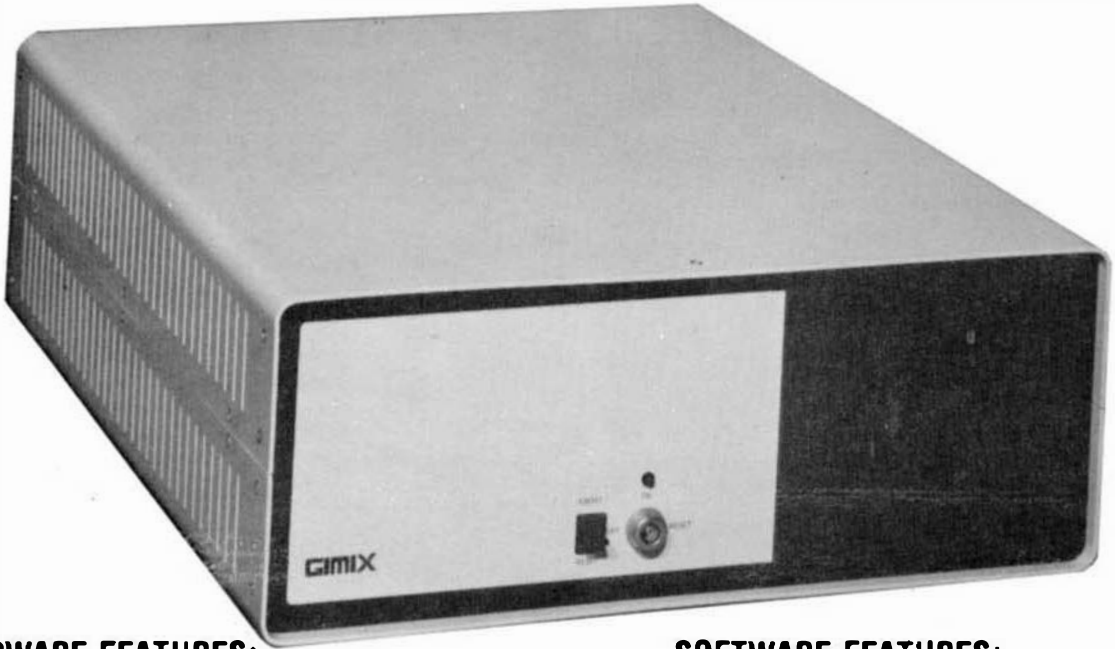
WESTCHESTER Applied Business Systems
Post Office Box 187, Briarcliff Manor, N.Y. 10510

All software is written in macro/assembler and runs under 6809 FLEX O/S. Terms: Check, Money Order, Visa or Mastercharge. Shipment first class. Add P&H $2.50 ($7.50 Foreign Surface or $15.00 Foreign Air). NY Res add sales tax. Specify 5" or 8".

Sales: S. E. MEDIA, (615) 842-4600, Consultation: 914-941-3552 (evens)

GMX 68020 DEVELOPMENT SYSTEM

A Multi-user, Multi-tasking software development system for use with all 68000 family processors.



HARDWARE FEATURES:

- The GMX-020 CPU board has: the MC68020 32-bit processor, a 4K Byte no wait-state instruction cache, high-speed MMU, and a full-featured hardware time of day clock/calendar with battery back-up. It also provides for an optional 68881 floating point co-processor.
- 1 Megabyte of high speed static RAM.
- Intelligent Serial and Parallel I/O Processor boards significantly reduce system overhead by handling routine I/O functions. This frees up the host CPU for running user programs. The result is a speed up of system performance and allows all terminals to run at up to 19.2K baud.
- The system hardware will support up to 39 terminals.
- Powered by a constant voltage ferro-resonant power supply that insures proper system operation under adverse AC power input conditions.
- DMA hard disk interface and DMA double density floppy disk controller are used for data transfers at full bus speed. The DMA hard disk drive controller provides automatic 22-bit burst data error detection and 11-bit burst error correction.
- A selection of hard disk drives with capacities from 19 to 85 Megabytes, removeable pack hard disk drives, streaming tape drives, and floppy disk drives is available.

SOFTWARE FEATURES:

The UniFLEX VM Operating System is a demand-paged, virtual memory operating system written in 68020 Assembler code for compactness and efficiency. Any UniFLEX system will run faster than a comparable system written in a higher level language. This is important in such areas as context switching, disk I/O, and system call handling. Other features include:

- compact, efficient Kernel and modules allows handling more users more effectively than UNIX systems, using much less disk space.
- UNIX system V compatibility at the C source code level.
- C Compiler optimized in 68020 code (optional).
- Record locking for shared files.
- Users can share programs in memory.
- Modeled after UNIX systems, with similar commands.
- System accounting facilities.
- Sequential and random file access.
- Maximum record size limited only by the disk size.
- Multiple Level Directories.
- Up to 4 Megabytes of Virtual Memory per user.
- Optional Languages available are: C, BASIC, COBOL, FORTRAN, LISP, PROLOG, SCULPTOR, and ASSEMBLER. In development are ADA, PASCAL, and FORTH.

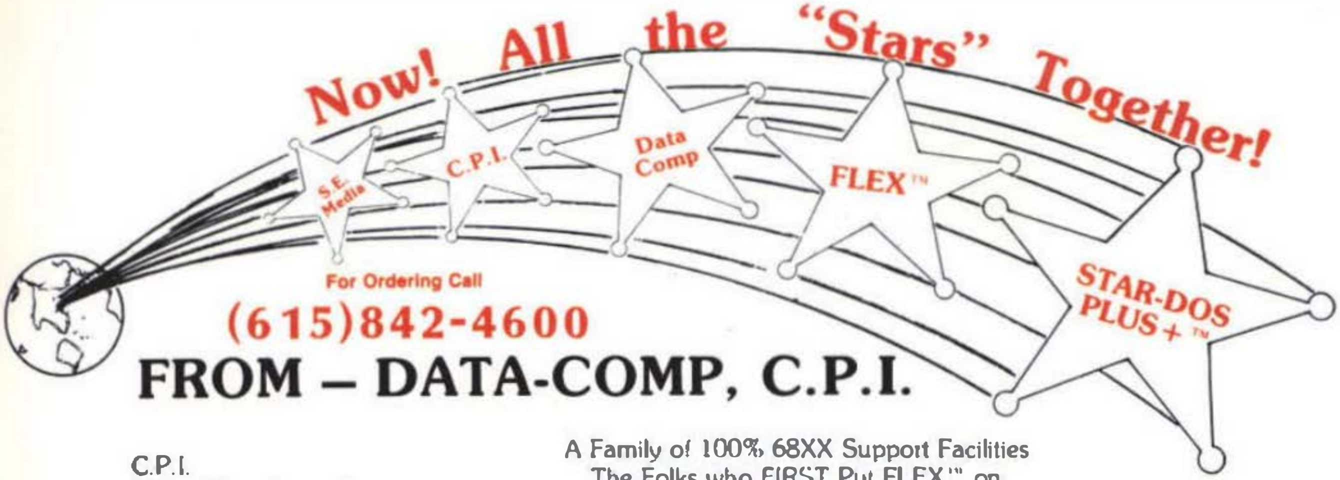
Included with the UniFLEX Operating System are a Utilities package, editor, relocating assembler, linking loader, and printer spooler. Options Inlcude a fast floating point package, library generator, and a sort-merge package.

The GMX version of the MOTOROLA 020 BUG is Included with the system.

UNIX is a trademark at A. T. & T.
ADA is a trademark of the U.S. Government
UniFLEX is a trademark of Technical Systems Consultants, Inc.
GMX and GIMIX are trademarks of GIMIX, Inc.

GIMIX, Inc., a Chicago based microcomputer company established in 1975, has produced state of the art microcomputer systems based on Motorola 6800 and 6809 microprocessors. GIMIX systems are in use in Industry,, Hospitals, Universities, Research Organiations, and by Software Developers. GIMIX was awarded the prestigious President's "E" Certificate for Exports in 1984.

GIMIX Inc. 1337 WEST 37th PLACE • CHICAGO, ILLINOIS 60609 • (312) 927-5510 • TWX910-221-4055



For Ordering Call

(615)842-4600

FROM – DATA-COMP, C.P.I.

C.P.I.
Color Micro Journal
'68' Micro Journal
Data-Comp
S.E. Media

A Family of 100% 68XX Support Facilities
The Folks who FIRST Put FLEX™ on
The CoCo
Now Offering: *FLEX™ (2 Versions)
AND *STAR-DOS PLUS+™

FLEX-CoCo Sr.
with TSC Editor
TSC Assembler
Complete with Manuals
Reg. $250.00 **Only $79.95**

STAR-DOS PLUS+
- Functions Same as FLEX
- Reads - writes FLEX Disks
- Run FLEX Programs
- Just type: Run "STAR-DOS"
- Over 300 utilities & programs to choose from.

$34.50

FLEX-CoCo Jr.
without TSC
Editor & Assembler
$49.95

PLUS

ALL VERSIONS OF FLEX & STAR-DOS• INCLUDE

+ Read-Write-Dir RS Disk
+ Run RS Basic from Both
+ More Free Utilities
+ Many Many More!!!
+ External Terminal Program
+ Test Disk Program
+ Disk Examine & Repair Program
+ Memory Examine Program

TSC Editor
Reg $50.00
NOW $35.00

TSC Assembler
Reg $50.00
NOW $35.00

CoCo Disk Drive Systems

NEW LOWER PRICES ON **PAK #5**, AND **PRINTERS**

THESE PACKAGES INCLUDE DRIVE, *CONTROLLER,
POWER SUPPLY & CABINET, CABLE, AND MANUAL.

* SPECIFY WHAT CONTROLLER YOU WANT **J&M**, OR **RADIO SHACK**.

PAK #1 - 1 SINGLE SIDED, DOUBLE DENSITY SYS.	$349.95
PAK #2 - 2 SINGLE SIDED, DOUBLE DENSITY SYS.	$639.95
PAK #3 - 1 DOUBLE SIDED, DOUBLE DENSITY SYS.	$439.95
PAK #4 - 2 DOUBLE SIDED, DOUBLE DENSITY SYS.	$699.95
PAK #5 - 2 DOUBLE SIDED, DOUBLE DENSITY SYS. THINLINE DRIVES, HALF SIZE	**$499.95**

Controllers

J&M DISK CONTROLLER W/ JDOS OR RADIO SHACK DISK BASIC, SPECIFY WHAT DISK BASIC.	$134.95
RADIO SHACK DISK CONTROLLER 1.1	$134.95

Misc.

64K UPGRADE W/MOD. INSTRUCTIONS, C,D,E,F, AND COCO 2	$ 44.95
HJL KEYBOARDS	$ 74.95
MICRO TECH LOWER CASE ROM ADAPTER	$ 74.95
RADIO SHACK BASIC 1.2	$ 24.95
RADIO SHACK DISK BASIC 1.1	$ 24.95
DISK DRIVE CABINET & POWER SUPPLY	$ 49.95
SINGLE SIDED, DOUBLE DENSITY 5" DISK DRIVE	$199.95
DOUBLE SIDED, DOUBLE DENSITY 5" DISK DRIVE	$249.95

Printers

EPSON RX-80	$269.00
EPSON RX-80FT	$369.00
EPSON MX-100	$499.00
EPSON FX-100	$799.00
EPSON FX-80	$549.00
EPSON MX-70	$200.00

Disk Drive Cables

CABLE FOR ONE DRIVE	$ 19.95
CABLE FOR TWO DRIVES	$ 24.95

DATA-COMP
5900 Cassandra Smith Rd.
Hixson, TN 37343

VISA

SHIPPING
USA ADD 2%
FOREIGN ADD 5%
MIN. $2.50

☎
(615)842-4600
For Ordering
TELEX 558 414 PVT BTH

68 Micro Journal
ISSN 0194-5025
5900 Cassandra Smith
Hixson, TN 37343

Second Class Postage Paid
At Hixson, TN
And Additional Mailing Offices